

SYNTHESIS, ANALYSIS, AND REVISION OF CORRECT-BY-CONSTRUCTION CONTROLLERS FOR ROBOTS WITH SENSING AND ACTUATION ERRORS

A Dissertation

Presented to the Faculty of the Graduate School
of Cornell University

in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

by

Benjamin Lee Johnson

January 2015

© 2015 Benjamin Lee Johnson
ALL RIGHTS RESERVED

SYNTHESIS, ANALYSIS, AND REVISION OF CORRECT-BY-CONSTRUCTION CONTROLLERS FOR ROBOTS WITH SENSING AND ACTUATION ERRORS

Benjamin Lee Johnson , Ph.D.

Cornell University 2015

Recently developed techniques in automatic synthesis of correct-by-construction robot controllers from a set of high-level task specifications offer a number of advantages over other, more traditional, methods of programming robots. Such synthesis techniques allow for quick, intuitive creation of controllers for complex tasks, such that the resulting controller is guaranteed to satisfy its underlying task specification. This guarantee, however, is predicated on the assumption that the robot operates without error in its sensing and actuation.

My Ph.D. dissertation provides methods for probabilistically modeling errors in the robot's sensors and actuators, and using those models to compute the probability that the robot (running such a synthesized controller) will exhibit the desired behavior. Furthermore, I provide methods for leveraging that analysis of the controller to automatically provide the user with suggestions for revisions to the task specification, in order to improve the robot's probabilistic behavior. Finally, I discuss the incorporation of the sensor/actuator error models into the actual synthesis step. By considering the error models, we can synthesize the controller that is most likely to satisfy the given specification, when operating with the modeled errors. This work provides an important tool for reliably applying controller synthesis techniques to real-world problems, where the robot's sensors and actuators are imperfect.

BIOGRAPHICAL SKETCH

Benjamin Johnson received his B.S. in Mechanical Engineering at the University of Idaho in 2009, graduating Suma Cum Laude. He also graduated with a minor in mathematics, and a certificate from the University of Idaho Honors Program. Ben received his M.S. degree in Mechanical Engineering from Cornell University in 2013, en route to earning his Ph.D. in 2015. His doctorate degree is in Mechanical Engineering, with a minor in Computer Science.

This work is dedicated to my friends and family. Their continual love and support means more to me than they could know.

ACKNOWLEDGEMENTS

The work contained in this thesis was supported by the following grants: NSF CNS-0931686 and DARPA N66001-12-1-4250.

I would like to acknowledge my collaborators and associates at Cornell University, who have helped in various ways with the research presented in this thesis. Cameron Finucane and Vasumathi Raman have provided valuable support, advice, and discussion for the majority of my tenure at Cornell University. Frank Havlak and Gangyuan Jing have both contributed to collaborative work, presented in this thesis. Eric Sample has assisted with implementation and deployment of work necessary for the completion of a presented experiment. Additionally, the entirety of the Verifiable Robotics research group at Cornell University have provided value discussions and advice.

I would also like to acknowledge my committee members Dr. Mark Campbell and Dr. Joseph Halpern, who have provided advice and guidance.

Most importantly, I want to acknowledge my advisor: Dr. Hadas Kress-Gazit. She has served as an advisor and role-model both professionally and personally, and I have been privileged to have spent the past years learning from her.

TABLE OF CONTENTS

1	Introduction	1
2	Probabilistic Guarantees for High-Level Robot Behavior in the Presence of Sensor Error	4
2.1	Introduction	4
2.2	Preliminaries	8
2.2.1	Linear Temporal Logic (LTL)	8
2.2.2	Labeled Transition Systems (LTS)	9
2.2.3	Discrete Time Markov Chains	9
2.2.4	Markov Decision Processes	10
2.3	Problem Statement	10
2.4	Approach	12
2.4.1	Building the DTMC	14
2.4.2	Parametric Model Checking	16
2.4.3	PRISM Non-Parametric Model Checking	20
2.5	Examples	21
2.5.1	Example 1: Housekeeping Robot	21
2.5.2	Example 2: Taxi	29
2.6	Conclusion	35
3	Analyzing and Revising Synthesized Controllers for Robots with Sensing and Actuation Errors	37
3.1	Introduction	37
3.2	Preliminaries	40
3.2.1	Linear Temporal Logic	40
3.2.2	Controller Synthesis	41
3.2.3	Probabilistic Model Checking	43
3.3	Problem Statement	43
3.4	Modeling and Analysis	45
3.4.1	Sensor Error	46
3.4.2	Actuator Error	49
3.4.3	Model Analysis	51
3.5	Revision Suggestion	51
3.5.1	Safety Specification Revisions	52
3.5.2	Initial-State Revisions	55
3.5.3	Low-Level Component Revisions	56
3.5.4	Liveness Condition Revisions	58
3.5.5	Notes on Revision Suggestions	59
3.6	Examples	60
3.7	Conclusion	67

4	Automatic Synthesis of High-Level Controllers for Robots With Imperfect Sensing and Actuation	68
4.1	Introduction	68
4.1.1	Related Work	69
4.1.2	Example 1	71
4.2	Preliminaries	72
4.2.1	Linear Temporal Logic	72
4.2.2	Controller Synthesis	73
4.2.3	Optimal Controller Synthesis	76
4.2.4	Probabilistic Error Models	77
4.3	Problem Statement	79
4.4	Cost Function	80
4.5	Results	86
4.5.1	Changing ψ	86
4.5.2	Changing the Model Probabilities	88
4.6	Conclusion	89
5	Experimental Evaluation and Formal Analysis of High-Level Tasks with Dynamic Obstacle Anticipation on a Full-Sized Autonomous Vehicle	92
5.1	Introduction	92
5.1.1	Related Work	95
5.2	Preliminaries	96
5.2.1	Linear Temporal Logic	96
5.2.2	Controller Synthesis	98
5.2.3	Gaussian Mixture Models	99
5.3	Technical Approach and Problem Formulation	100
5.3.1	Scenario	101
5.3.2	Synthesized Controller	103
5.3.3	Dynamic Obstacle Anticipation	104
5.3.4	Sensor Abstraction	106
5.4	Analysis	108
5.5	Experimental Platform	110
5.5.1	Autonomous Car	110
5.5.2	Virtual Obstacles	113
5.6	Results	113
5.6.1	Formal Analysis	115
5.6.2	Experiment	120
5.6.3	Results comparison	122
5.7	Conclusions	124

CHAPTER 1

INTRODUCTION

The increasingly common-place and complex use of autonomous systems has created a need for techniques that allow for the rapid creation and deployment reliable robot controllers. That is: one should be able to quickly and easily create or modify robot controllers in such a way that the controller will operate in the intended manner. To this end, techniques such as [37] use formal methods to automatically synthesize a correct-by-construction robot controller from a higher-level task specification. Such methods ease the technical burden of creating a complex controller, and do so in a manner that provides guarantees on the resulting behavior of the robot. Such guarantees commonly make the assumption that the robot operates without errors in its sensing and actuation; the work presented in this thesis relaxes that assumption, and investigates the probabilistic analysis, synthesis, and revision of such controllers.

Chapter 2 of this thesis presents an approach for reasoning about the effects of sensor error on high-level robot behavior. It considers robot controllers that are synthesized from high-level, temporal logic task specifications such that the resulting robot behavior is guaranteed to satisfy these specifications when assuming perfect sensors and actuators, and relaxes the assumption of perfect sensing. The behavior of the system is then modeled probabilistically, and the probability with which the controller satisfies a set of temporal logic specifications can be calculated using model checking techniques. This chapter considers both parametric representations, where the satisfaction probability is found as a function of the model parameters, and numerical representations, allowing for the analysis of large examples. It also considers models in which some parts of the environment and sensor have unknown transition probabilities, and describes a method to determine upper and lower bounds for the probability. The approach is illustrated with two examples that provide insight into unintuitive effects of sensor error that can

inform the specification design process.

Chapter 3 of this thesis relaxes the assumption on both sensing and actuation, and describes a method for probabilistically analyzing the behavior of a robot controller that is synthesized from a set of temporal logic specifications, when the robot operates with uncertainty in its sensing and actuation. The described approach creates a probabilistic model of the system and uses probabilistic model checking techniques to find the probability that it satisfies some set of specifications. Additionally, this chapter presents a method which leverages that analysis to provide automated feedback to the user in the form of suggested revisions to the task specification or low-level components, in order to increase the probability that the robot successfully accomplishes its task.

The automatic synthesis of robot controllers from high-level specifications offers a number of advantages over more traditional programming approaches. One such advantage is that the synthesized controller is verifiably correct; that is, it is guaranteed to satisfy all of its underlying specifications given the assumption of perfect sensing and actuation. The work presented in Chapter 4 relaxes this assumption, and synthesizes a discrete controller that minimizes a cost function which captures the probability that the robot will violate a given temporal logic formula, while constrained by the original task specification. This chapter discusses the cost function and controller synthesis, and illustrates the approach with an example problem.

Chapter 5 fuses the probabilistic results of dynamic obstacle anticipation with the deterministic decision making of a formally synthesized robot controller running on a full-sized autonomous vehicle. The obstacle anticipation (used to calculate the probability of collision with dynamic obstacles around the vehicle) is abstracted to a set of Boolean observations, which are then used by the synthesized controller (a state machine generated from temporal logic task specifications). The obstacle anticipation, sensor abstraction, and synthesized controller are implemented on a full-scale autonomous

vehicle, and experimental data is collected and compared to a formal analysis of the probabilistic behavior of the system. A comparison of the results shows good agreement between the formal analysis and the experimental results.

CHAPTER 2

PROBABILISTIC GUARANTEES FOR HIGH-LEVEL ROBOT BEHAVIOR IN THE PRESENCE OF SENSOR ERROR

2.1 Introduction

The creation of robot controllers for complex tasks is an arduous and error-prone process, requiring the iterative design and testing of large, complex controllers. One increasingly popular approach is to employ hybrid controllers that allow for discrete switching among a set of individual, continuous controllers. Such controllers are, however, difficult and time-consuming to create and test. One method to address this problem is through the automated creation and verification of controllers. Recently, researchers have developed methods for automatically synthesizing complex, hybrid controllers from high-level task specifications in a manner that provides guarantees about the behavior of the robot [4, 8, 17, 34, 36, 37, 44, 55, 56, 62].

Such methods use temporal logic formulas to specify a variety of complex behaviors (such as “go to the *bedroom*, but avoid the *kitchen*” or “if sensing a *person*, then *radio* for help”), and cover a variety of capabilities and applications. The approaches presented in [34] and [32], for example, facilitate the creation of non-reactive controllers (i.e. the behavior does not depend on changes in the environment) for complex motion planning. Conversely, [37] and [62] focus on creating controllers that react to the system’s perception of environmental inputs. Another example of differing approaches is that of [16] and [61] when compared to [4] and [32]; the former two methods use feedback controllers, while the latter use sampling-based methods to allow for motion planning involving complex dynamics and environments, at the expense of completeness.

Previous work facilitated the creation of controllers that were guaranteed to satisfy their underlying specifications, but such guarantees were based on the assumption of

perfect sensing and actuation; that is, such controllers assumed that the robot knows the exact state of the environment in which it operates, and that the outcomes of all robot actions are deterministic. While we maintain the assumption of perfect actuation, in this paper, we relax the assumption of perfect sensing, incorporating uncertainty into the sensor abstractions. After creating a model of the system, we use probabilistic analysis to investigate the correctness of the high-level behavior of the controllers.

Related work includes [42], in which the authors control linear, stochastic systems with temporal logic specifications by constructing a Markov Decision Process (MDP) and using model checking algorithms to find an execution satisfying the specifications. Building on the work presented in [34], they then define a sequence of controllers to maximize the probability of following the execution obtained from the model checking algorithms.

Similarly, in [43] the authors propose an algorithm for finding a desirable control strategy for motion planning in the presence of noise. By treating the outcome of the low-level motion controllers (i.e. the transitions between regions) in a probabilistic fashion, the authors were able to find, under known operating conditions, the control strategy most likely to satisfy the given specifications. They consider scenarios in which actuator uncertainty and uncertainty in the continuous sensor (i.e. location within a specified region), result in probabilistic transitions between states. The sensor abstraction (i.e. which discrete region the robot is in), however, is still assumed to be perfect.

The work presented in [10] extends this approach, allowing for the synthesized control strategy to react to time-varying observations of the environment. These observations are modeled as independent and identically distributed probabilities, and the authors present an approach for generating a reactive control strategy that maximizes the probability that the robot satisfies a given temporal logic formula.

In this paper, we assume perfect actuation, and analyze the effect of sensor false-

positives and false-negatives. Specifically, we model the system as either a Discrete-Time Markov Chain (DTMC) or an MDP, and use probabilistic model-checking techniques to perform an offline analysis of the robot’s satisfaction of behavioral specifications defined as temporal logic formulas. In the case where the behavior of the environment and sensors can be models with a set of known probabilities, the system is modeled as a DTMC, and an expected satisfaction probability can be found. If the event frequencies or sensor accuracies are unknown, the system is modeled as an MDP, and bounds are found for the minimum and maximum probability that a specification is satisfied. In both cases, the analysis provides insights into the effects of specific sensor errors on the high-level specification guarantees, and can be used to inform adjustments to the specifications, improvements to the sensors, and changes to the controller, in an effort to improve the robot’s performance.

We consider a parametric model checking algorithm, based on the algorithm described in [22]. This algorithm calculates the probability with which the system model satisfies a set of temporal logic formulas, as a function of the model parameters; obtaining such a function allows for rapid evaluation of the performance of the system given known sensor error, or the calculation of bounds on sensor accuracy for a desired performance level. Models with large state-spaces and many transitions can become intractable for our current implementation of the algorithm; for these cases, we use the PRISM [41] probabilistic model checking tool with the same system model and specifications to find a numerical probability for the satisfaction of the specifications. The key difference between the two techniques is that the parametric model checking algorithm computes a formula for the probability, as a function of the system parameters (sensor error and environment event frequencies), while PRISM is used to evaluate the probability at a specific valuation of the parameter; while the initial computational cost of the parametric formula is higher, individual evaluations of the formula are computationally

very cheap.

The work presented in the paper is, to the best of our knowledge, novel; it addresses a different but related problem to that addressed in the most closely related literature: [42, 43], and [10]. In [42, 43], the authors present a method for synthesizing a non-reactive controller that maximizes the probability of satisfying a temporal logic formula. In this work, we provide a method for analyzing the effects of erroneous high-level sensing on the behavior of a reactive controller. Analysis of the controller after the synthesis stage provides insight into specific sources of error; such insights enable targeted improvements of the controller and sensors. The presented work also differs from that presented in [10], which allows for the observation of probabilistic properties of the environment but does not investigate the truth of those observations (i.e. the uncertainty is in the outcome of the robot’s motion primitives, not in the accuracy of the high-level observations).

The research and results presented in this paper are a continuation of those presented in [28]. Most notably, this paper adds the capability of using MDPs to model and analyze a system in which the environment model is not precisely known (i.e. event frequencies are unknown). In either case, it is important that the modeled probabilities accurately capture the characteristics of the system being modeled. In [27] the authors apply this approach to the problem of an autonomous car navigating an urban environment; they anticipate the behavior of dynamic obstacles and abstract the anticipated probability of collision to a binary value representing the safety of different maneuvers at intersections. The authors then use Monte-Carlo simulation statistics to model the sensor error and environment behavior, and evaluate the safety of the vehicle. The results obtained from the formal analysis compare favorably with those found in simulation.

The paper is organized as follows. Section 2.2 briefly describes the background information, and Section 2.3 formally defines the problem. Section 2.4 describes our

approach and the algorithms we use to analyze the effect of sensor error. Section 2.5 presents two illustrative examples and their results. Conclusions are given in Section 2.6.

2.2 Preliminaries

2.2.1 Linear Temporal Logic (LTL)

The syntax of Linear Temporal Logic (LTL) is defined using a set of atomic propositions ($\pi \in \Pi$), the set of boolean operators (“not”: \neg , “and”: \wedge), and the set of temporal operators (“next”: \bigcirc , “until”: \mathcal{U}). The syntax for the language is then defined recursively as follows.

$$\phi ::= \text{true} \mid \pi \mid \neg\phi \mid \phi \wedge \phi \mid \bigcirc\phi \mid \phi \mathcal{U}\phi$$

We can define the additional boolean operators \vee (“or”), \rightarrow (“implies”) and \leftrightarrow (“if and only if”) using \neg and \wedge . Likewise, \bigcirc and \mathcal{U} can be used to define the operators \Diamond (“eventually”) and \Box (“always”).

To define the semantics of LTL, let ω represent an infinite sequence of states in a transition system [14]. Let $\omega_i \subseteq 2^\Pi$ be the labeling of the state at position i in the infinite sequence ω with a subset of Π that is true at time i . Intuitively, π is true iff $\pi \in \omega_i$, and $\bigcirc\phi$ is true if the formula ϕ is true in the next step (at state ω_{i+1}). The formula $\Diamond\phi = \text{true}\mathcal{U}\phi$ is true if ϕ holds in at least one state in ω , while $\Box\phi = \neg\Diamond\neg\phi$ is true if ϕ holds for all states in ω . For a formal definition of the semantics of LTL, the reader is referred to [14].

We consider a restricted class of LTL formulae [51] of the form $\varphi = (\varphi_e \rightarrow \varphi_s)$. The formula φ represents the desired robot dynamics φ_s for an environment φ_e ; the formula φ_e also serves to place constraining assumptions on the behavior of the environment.

We assign φ_e and φ_s the following structure.

$$\varphi_e = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e$$

$$\varphi_s = \varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s$$

In the preceding formulas, φ_t^s represents the **safety** requirements for the robot, which require certain behaviors to always be satisfied (e.g. “always *stop* when sensing a *red light*”). It consists of a conjunction of formulas of the form $\Box B_i$, where each B_i is a Boolean formula over $\Pi \cup \bigcirc\Pi$ (e.g. $\Box(\bigcirc RedLight \rightarrow \bigcirc Stop)$). φ_g^s represents the specified **liveness** for the robot, which requires that some desirable behavior occurs infinitely often (e.g. “while not *stopped*, repeatedly visit the *parking lot*”). It consists of a conjunction of formulas of the form $\Box\Diamond B_j$, where each B_j is a boolean formula over Π (e.g. $\Box\Diamond(\neg Stop \rightarrow ParkingLot)$). The remaining portion of φ_s (φ_i^s) represents the set of initial conditions for the robot, and the components of φ_e represent the initial (φ_i^e), safety (φ_t^e), and liveness (φ_g^e) assumptions on the environment. For more information, see [51].

2.2.2 Labeled Transition Systems (LTS)

A Labeled Transition System (LTS) is a tuple $\mathcal{A} = \{S, S_0, \Sigma, \delta, \Pi, L\}$ where S is a set of states, S_0 is the set of initial states and Σ is the input alphabet. The transition relation $\delta : S \times 2^\Sigma \rightarrow S$ defines the possible labeled transitions (s_i, σ_{ij}, s_j) for $\sigma_{ij} \subseteq \Sigma$. Π is the set of atomic propositions labeling the states, and the function $L : S \rightarrow 2^\Pi$ labels each state with a set of symbols belonging to the set Π .

2.2.3 Discrete Time Markov Chains

A Discrete Time Markov Chain (DTMC) is defined as a tuple $\mathcal{D} = \{Q, Q_0, \Delta, \Pi, \mathcal{L}\}$, where Q defines a finite set of states with the set of initial states Q_0 . The transition

function $\Delta : Q \times Q \rightarrow P$ defines the probability $p \in P$ with which state $q_i \in Q$ transitions to state $q_j \in Q$, where $P = (0, 1]$. Π is the set of atomic propositions labeling the states, and the function $\mathcal{L} : Q \rightarrow 2^\Pi$ labels each state with a set of symbols belonging to the set Π .

2.2.4 Markov Decision Processes

Markov Decision Processes (MDPs) are an extension of DTMCs, which include the labeling of the transitions, allowing for nondeterministic choices. An MDP is defined as a tuple $\mathcal{M} = \{Q, Q_0, \Sigma, \Delta_{MDP}, \Pi, \mathcal{L}\}$ where Q defines a finite set of states with the set of initial states Q_0 . The transition function $\Delta_{MDP} : Q \times 2^\Sigma \times Q \rightarrow P$ defines the probability $p \in P$ of transitioning from state $q_i \in Q$ to state $q_j \in Q$, given the input $\sigma_{ij} \subseteq \Sigma$, where $P = (0, 1]$. Π is the set of atomic propositions labeling the states, and the function $\mathcal{L} : Q \rightarrow 2^\Pi$ labels each state with a set of symbols belonging to the set Π .

2.3 Problem Statement

We consider automatically-synthesized robot controllers (e.g. [37]) for which the robot is guaranteed to achieve a given high-level specification, if perfect sensing and actuation are assumed. To analyze the behavior of the controller under sensor error, we define models for the robot controller, the environment behavior and the sensor error.

The robot controller, R , is defined as an LTS where the input set Σ is a set of binary propositions $\bar{X} = \{\bar{x}_1, \dots, \bar{x}_m\}$ that represent the information that the robot can attain about the environment via its sensors. $Y = \{r_1, \dots, r_n, a_1, \dots, a_k\}$ is the set of binary robot propositions (Π) used to label the states. The set of propositions $\{r_1, \dots, r_n\}$ represents the location of the robot in a partitioned environment, where proposition r_i is true if and only if the robot is in region i . The set $\{a_1, \dots, a_k\}$ is the set of propositions

that can be activated by the robot (e.g. the action *Stop*), where proposition a_j is true if and only if the robot is performing action j . The controller synthesis defines the possible states of the system S , the initial states S_0 , the transition function δ , and the labeling function L .

The environment behavior, E , is captured by a specified set of transition probabilities $P(X'|X, Y)$ defining the probability of the next environment proposition values $X' = \{x'_1, \dots, x'_m\}$, given the current environment and robot values, X and Y respectively. This formulation allows us, if applicable, to place assumptions on the behavior of the environment (by setting transition probabilities to 0 or 1). For example, we may restrict the value of *RedLight* to be false in non-intersection regions, resulting in a 0 probability transition to a state in which *RedLight* is true and the robot is not in an *Intersection*. Such assumptions enable us to generate controllers that do not need to satisfy the desired specification for any arbitrary environment, but rather only for a restricted set of allowable environments. The environmental assumptions, which restrict the values of X , are captured in the temporal logic formula φ_e .

The sensor behavior, \bar{E} , is modeled with the set of probabilities $P(\bar{X}'|X', \bar{X}, Y)$ defining the values of the next sensor propositions $\bar{X}' = \{\bar{x}'_1, \dots, \bar{x}'_m\}$ given the next environment values, X' , and the current proposition values of the sensors and the robot, \bar{X} and Y respectively. Such a model allows for the presence of false negatives ($\bar{x}' = False, x' = True$) and false positives ($\bar{x}' = True, x' = False$) in the abstracted sensor values.

In the case where either the environment or sensor behavior is not completely known (i.e. we do not have all of the transition probabilities for E or \bar{E} , respectively), we can instead model the behavior as an MDP. In this case, the transitions consist of a non-deterministic choice corresponding to the proposition(s) for which the transition probabilities are unknown. Propositions with known transition probabilities are modeled

within each non-deterministic choice.

The final piece we define is the system specification, Φ , for the robot. It is expressed as a set of LTL formulas $\{\phi_1, \dots, \phi_l\}$ that are part of the temporal logic formula φ_s [37]. In particular, we are interested in analyzing the behavior of the robot with respect to the safety and liveness constraints (φ_t^s and φ_g^s , respectively).

Previous work has shown that, given assumptions on the environment behavior and specifications for the robot behavior, a controller can be synthesized, such that the robot is guaranteed to behave correctly in all admissible environments (i.e. $E \parallel R \models \phi, \forall \phi \in \Phi$), should such a controller exist [37]. This guarantee, however, can only be made when perfect sensors and actuators are assumed (i.e. $E = \bar{E}$).

The focus of this paper is on the following problem.

Problem: Given a model of the robot controller R , and models of the environment E and sensors \bar{E} , determine the probability that the synthesized robot controller will satisfy a set of high level specifications when the sensor outputs contain false positives and false negatives. That is, given that $E \neq \bar{E}$, find $p(E \parallel \bar{R} \models \phi) \forall \phi \in \Phi$ where $\bar{R} = \bar{E} \parallel R$. For a model of the system where the environment or sensor model is incomplete (i.e. an MDP), find the upper and lower bounds on the probabilities, p_{min} and p_{max} .

2.4 Approach

To analyze the behavior of a synthesized controller (R) under sensor error, we compose it with models of the environment (E) and sensors (\bar{E}), and use probabilistic model checking techniques on the resulting model (DTMC or MDP) to assess the performance of the controller with respect to a set of LTL formulas. Figure 2.1 is a graphical overview of this process.

In the following, we describe an algorithm for creating a DTMC representing the

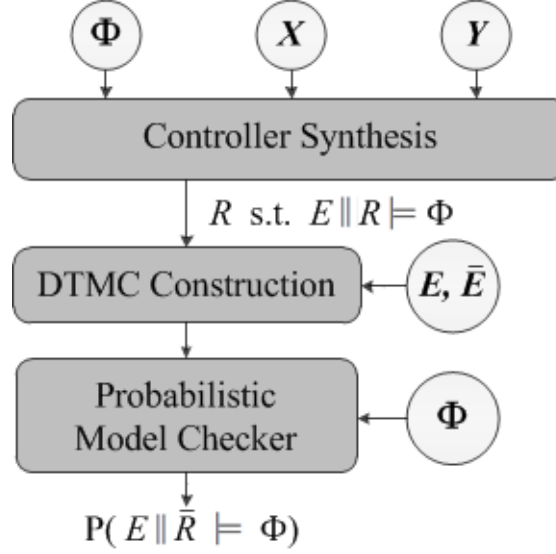


Figure 2.1: A flow chart of the approach presented in this paper for analyzing the correctness of a controller that is generated from high-level task specifications.

composition of the controller with the environment and sensor model and discuss two approaches to analyzing the robot’s behavior. The primary method we use is a parametric model checking algorithm, based on the work described in [22]. This algorithm returns the probabilities as rational functions over a set of variables, in our case, representing the environmental event frequency ($P(X'|X, Y)$) and sensor error rates ($P(\bar{X}'|X', \bar{X}, Y)$). Thus, we are able to provide information not only on the probability a specification is satisfied given the sensor error but also on the sensor error bounds that are required for a given desired specification probability.

In addition to the parametric model checking algorithm, we discuss the use of the off-the-shelf probabilistic model checker PRISM [39, 41] to evaluate probabilities when numeric model parameters are given (non-parametric). Additionally, this tool supports the analysis of systems that are modeled as MDPs. Such analysis allows us to find bounds on the probability for models in which the behavior of the environment or sensors cannot be fully characterized.

2.4.1 Building the DTMC

Analysis of a synthesized controller using the probabilistic model checking algorithm requires first that a model of the complete system be composed from the known models of the environment, sensors, and robot automaton. This process is described in Algorithm 1. The inputs are the LTS for the robot controller, the set of atomic propositions $\Pi = X \cup \bar{X} \cup Y$, the environment transition probabilities $P(X'|X, Y)$ and the sensor transition probabilities $P(\bar{X}'|X', \bar{X}, Y)$.

Given these inputs, the algorithm defines a set of probabilistic states for each deterministic state $s_i \in S$, such that the new probabilistic state is labeled with the combination of the environment propositions $X^j = (x_1, \dots, x_n) \in 2^X$, the sensor propositions \bar{X}^i , and the labels of the deterministic robot state (lines 3-6). The set of sensor propositions \bar{X}^i is defined by the labels on the transition (in the controller LTS) into the deterministic state s_i , from any predecessor s_h . Any probabilistic state created from $s_i \in S_0$ is then added to the set of initial states for the complete system (lines 7-8). The probabilistic transition function $\Delta : Q \times Q \rightarrow \mathcal{F}_V$ is then defined for all pairs of states (q_{ij}, q_{kl}) such that the underlying deterministic pair (s_i, s_k) is in the transition function δ , and the probability function $f_{ijkl} \in \mathcal{F}_V$ (the rational function representing the probability of transitioning from state q_{ij} to q_{kl}) is non-zero (lines 9-14). The notation in this algorithm uses the indices i and k to refer to individual states in the LTS for the synthesized controller, and the indices j and l to indicate different configurations of the environment; the state q_{ij} in the DTMC then refers to the state obtained by combining the controller state s_i with the sensor inputs \bar{X}^i and the environment configuration X^j . The resulting DTMC $\mathcal{D} = \{Q, Q_0, \Delta, \Pi, \mathcal{L}\}$ can then be used to analyze the performance of the controller.

To illustrate this process, consider a simple robot controller with one sensor \bar{x} . The controller enters state s_1 when the sensor proposition \bar{x} is true. Due to the presence

Algorithm 1 Define DTMC

```

1: procedure PROBDTMC( $\mathcal{A} = \{S, S_0, \bar{X}, \delta, Y, L\}$ ,  $\Pi = X \cup \bar{X} \cup Y$ ,
    $P(X'|X, Y), P(\bar{X}'|X', \bar{X}, Y)$ )
2:    $Q = \emptyset, Q_0 = \emptyset$ 
3:   for  $s_i \in S$  do
4:     for  $X^j \in 2^X$  do
5:        $Q = Q \cup q_{ij}$ 
6:        $\mathcal{L}(q_{ij}) = X^j \cup \bar{X}^i \cup L(s_i)$  s.t.  $(s_h, \bar{X}^i, s_i) \in \delta$ 
7:       if  $s_i \in S_0$  then
8:          $Q_0 = Q_0 \cup q_{ij}$ 
9:       for  $(s_i, \bar{X}^k, s_k) \in \delta$  do
10:         $Y^i = L(s_i)$ 
11:        for  $(q_{ij}, q_{kl}) \in Q \times Q$  do
12:           $f_{ijkl} = P(X^l|X^j, Y^i) \times P(\bar{X}^k|X^l, \bar{X}^i, Y^i)$ 
13:          if  $f_{ijkl} \neq 0$  then
14:             $\Delta = \Delta \cup (q_{ij}, f_{ijkl}, q_{kl})$ 
15:   return  $\mathcal{D} = \{Q, Q_0, \Delta, \Pi, \mathcal{L}\}$ 

```

of false-positives, however, this sensor proposition may be true when the environment proposition x is false (as well as when x is true). From the state s_1 , the algorithm creates 2 separate states for the DTMC: q_{11} (where x is true) and q_{12} (where x is false). In both states, the sensor proposition \bar{x} is true, and they share the labels of the controller state s_1 . A second controller state s_2 similarly results in the states q_{21} and q_{22} Which share the labels of s_2 and $\bar{x} = false$. The algorithm then calculates the transition probabilities for all of the state pairs from q_{11} or q_{12} to q_{21} or q_{22} . If the transition probability is non-zero, the state-probability-state tuple is added to the transition function Δ .

Additionally, we can define a bounded system by unfolding the DTMC found in Algorithm 1 over the range of time $0 \leq t \leq T$ where T is the desired time bound. Enforcing a bound on the system allows for the analysis of a finite execution of the robot, similar to performing bounded model checking on the DTMC. This process is described in Algorithm 2. Intuitively, this process can be seen as building a tree such that root nodes are defined as the set of initial states Q_0 , and each new level (referring to the next time step) is defined as the union of the sets of successors for each state in the preceding

level of the tree. Each state $q_{i,t}$ in the bounded system encodes both the discrete state q_i in the unbounded DTMC, and the respective time step t . This process is repeated for each time step $t \leq T$, and the resulting DTMC $\mathcal{D}_{bound} = \{Q_{bound}, Q_0, \Delta_{bound}, \Pi, \mathcal{L}, V\}$ is the bounded system.

Algorithm 2 Bound DTMC

```

1: procedure BOUNDSYS( $\mathcal{D} = \{Q, Q_0, \Delta, \Pi, \mathcal{L}, V\}, T$ )
2:    $Q_{prev} = Q_0$ 
3:    $Q_{bound} = \{q_{i,0} \mid \exists i \text{ s.t. } q_i \in Q_0\}$ 
4:    $Q_{0_0} = Q_{bound}$ 
5:    $t = 1$ 
6:   while  $t \leq T$  do
7:      $Q_{new} = \emptyset$ 
8:     for  $q_i \in Q_{prev}$  do
9:       for  $(f_{ij}, q_j) \text{ s.t. } (q_i, f_{ij}, q_j) \in \Delta$  do
10:         $Q_{new} = Q_{new} \cup q_j$ 
11:         $Q_{bound} = Q_{bound} \cup q_{j,t}$ 
12:         $\mathcal{L}(q_{j,t}) = \mathcal{L}(q_j)$ 
13:         $\Delta_{bound} = \Delta_{bound} \cup (q_{i,t-1}, f_{ij}, q_{j,t})$ 
14:      $Q_{prev} = Q_{new}$ 
15:      $t = t + 1$ 
16: return  $\mathcal{D}_{bound} = \{Q_{bound}, Q_{0_0}, \Delta_{bound}, \Pi, \mathcal{L}, V\}$ 

```

2.4.2 Parametric Model Checking

Given a probabilistic system model as defined in Section 2.4.1, we find a formula representing the probability with which it satisfies a specification of the form $\phi = \Diamond\varphi$ (resp. $\phi = \Box\varphi = \neg\Diamond\neg\varphi$), as a function of the system parameters. For the latter case, we find the probability of always satisfying φ by calculating the probability of eventually satisfying $\neg\varphi$, and subtracting it from 1 (giving us the probability of not eventually satisfying $\neg\varphi$). This process is described in Algorithm 3, and is based on the algorithm in [22]. Intuitively, lines 2-5 of the algorithm find the set of states $B \subseteq Q$ that satisfy the formula φ (resp. $\neg\varphi$). From this set of target states, we use a minimum fixed-point

operation to reduce the model to the set of states from which at least one target state is reachable (line 6). Line 7 then calls Algorithm 5, to find the probability of reaching the set of target states from an initial state. The function \mathcal{P} (resp. $1 - \mathcal{P}$) represents the probability that the formula $\phi = \Diamond\varphi$ (resp. $\phi = \Box\varphi$) is satisfied by the DTMC.

Algorithm 3 Parametric Model Checking

```

1: procedure PARAMETRICMC( $\mathcal{D} = \{Q, Q_0, \Delta, \Pi, \mathcal{L}\}, \phi, X, Y$ )
2:   if  $\phi = \Box\varphi$  then
3:      $(B, \Delta) = \text{TargetStates}(Q, \Delta, \neg\varphi, \mathcal{L}, X, Y)$ 
4:   else
5:      $(B, \Delta) = \text{TargetStates}(Q, \Delta, \varphi, \mathcal{L}, X, Y)$ 
6:    $Q_{\text{reach}} = \text{ReduceStates}(Q, B, \Delta)$ 
7:    $\mathcal{P} = \text{Eliminate}(Q_{\text{reach}}, Q_0, B, \Delta)$ 
8:   if  $\phi = \Box\varphi$  then
9:      $\mathcal{P} = 1 - \mathcal{P}$ 
10:  return  $\mathcal{P}$ 

```

Algorithm 4 defines the process of finding the set of states that satisfy a given formula ϕ . To do this, we look at two distinct forms of ϕ . In the first case, ϕ is a boolean formula over the values of X and Y in a single state (in this paper, we examine the performance of the controller with respect to the environment, and do not use \bar{X} in any of the formulas). For such a formula, the set of target states can be found by looping through all of the states in the model. At each state, we map each atomic proposition in the formula to a T/F value, based on the state label (line 6). If the resulting formula evaluates to true, we add that state to the set of target states and remove all transitions out of it, treating it as a sink (lines 7-9).

The second case is that for which ϕ is a boolean formula over propositions in the current and next state ($X, Y, \bigcirc X, \bigcirc Y$). For this case, we create a single goal state q^* and loop through each pair of states (q_i, q_j) where q_j is a successor of q_i . We then map each atomic proposition to a T/F value based on the state q_j if it is under the scope of a “next” operator (line 18), or based on the state q_i otherwise (line 17). If the formula is true when evaluated over the pair, the transition from q_i to q_j is removed and the

Algorithm 4 Find Target States

```

1: procedure TARGETSTATES( $Q, \Delta, \phi, \mathcal{L}, X, Y$ )
2:   if  $\phi$  is a formula over  $\{X, Y\}$  then
3:      $B = \emptyset$ 
4:     for  $q_i \in Q$  do
5:       for  $a \in \{X, Y\}$  do
6:          $\phi[a \mapsto (a \in \mathcal{L}(q_i))]$ 
7:         if  $eval(\phi) == \text{True}$  then
8:            $B = B \cup q_i$ 
9:            $\Delta = \Delta \setminus (q_i, f_{ij}, q_j) \quad \forall q_j \in post(q_i)$ 
10:    else if  $\phi$  is a formula over  $\{X, Y, \bigcirc X, \bigcirc Y\}$  then
11:       $Q = Q \cup q^*$ 
12:       $B = \{q^*\}$ 
13:      for  $q_i \in Q \setminus q^*$  do
14:         $f_{i*} = 0$ 
15:        for  $q_j \in post(q_i)$  do
16:          for  $a \in \{X, Y\}$  do
17:             $\phi[a \mapsto (a \in \mathcal{L}(q_i))]$ 
18:             $\phi[\bigcirc a \mapsto (a \in \mathcal{L}(q_j))]$ 
19:            if  $eval(\phi) == \text{True}$  then
20:               $f_{i*} = f_{i*} + f_{ij}$ 
21:               $\Delta = \Delta \setminus (q_i, f_{ij}, q_j)$ 
22:          if  $f_{i*} \neq 0$  then
23:             $\Delta = \Delta \cup (q_i, f_{i*}, q^*)$ 
24:    return  $B, \Delta$ 

```

transition probability is added to the transition from q_i to q^* (lines 19-21).

Given the set of initial states Q_0 , the set of goal states B , the set of states Q_{reach} that can reach B , and the transition function Δ , we can now eliminate all intermediate states from the model, until we are left with only the initial states, the goal states, and the associated transition probabilities. Algorithm 5 describes this process, and the calculation of the probability with which our model will reach a target state from a specified initial state. This algorithm follows the process described in [22], and can intuitively be described as removing an intermediate state q_i and accounting for the probability with which a predecessor $q_1 \in Pre(q_i)$ will transition through q_i to a successor $q_2 \in Post(q_i)$, for all pairs (q_1, q_2) (lines 5-9). Once all intermediate states are eliminated, the probability with which the system will reach a state satisfying ϕ is the sum of transition probabilities from the initial state to each goal state (adjusting for self-transitions) (line 10).

Algorithm 5 State Elimination

```

1: procedure ELIMINATE( $Q_{reach}, Q_0, B, \Delta$ )
2:   for  $q_0 \in Q_0$  do
3:      $Q_{elim} = Q_{reach} \setminus \{B, q_0\}$ 
4:     for  $q_i \in Q_{elim}$  do
5:       for  $(q_1, q_2) \in Pre(q_i) \times Post(q_i)$  do
6:          $f_{12} = f_{12} + f_{1i} \frac{1}{1-f_{ii}} f_{i2}$ 
7:          $(q_1, f_{12}, q_2) \in \Delta$ 
8:          $\Delta = \Delta \setminus \{(q_1, f_{1i}, q_i), (q_i, f_{i2}, q_2)\}$ 
9:        $Q_{elim} = Q_{elim} \setminus q_i$ 
10:     $\mathcal{P}(q_0, B) = \frac{1}{1-f_{00}} \sum_{b \in B} f_{0b}$ 
11:  return  $\mathcal{P}$ 

```

Our implementation of the preceding algorithms is done in Python, using the SymPy-Core toolbox [49] to perform the symbolic manipulations. This prototypical implementation is limited primarily by the number of transitions in the composed system model (though it is also affected by other factors such as the number of states or the length of the LTL formula); we found that it can reasonably handle systems with roughly 3,000

transitions, but encounters computational issues with larger models. A more optimized implementation would likely alleviate many of these issues, and allow for faster calculations of the parametric formulas. The theoretical complexity of the algorithm is linear in the length of the formula ϕ in Algorithm 4 [40], and cubic in the number of transitions to be eliminated in Algorithm 5 [22].

2.4.3 PRISM Non-Parametric Model Checking

In place of the Parametric Model Checking algorithm discussed above, we can also use the off-the-shelf model checking software PRISM [41], with the same system model and specifications, to analyze the performance of the controller. The PRISM software allows for calculating the probability that the system model satisfies an LTL formula, but it does not use parametric algorithms, restricting the evaluation to specific numerical valuations of the system model. In particular, we use PRISM to evaluate models that are too large for our current implementation of the parametric algorithm. The theoretical complexity of LTL model checking is, in general, exponential in the size of the LTL formula (though in our application the size of the LTL formula tends to be reasonably small), and linear in the size of the system model [53].

In addition to being able to analyze the probability that a particular DTMC satisfies an LTL specification, PRISM can also be used to analyze the minimum and maximum probabilities that a specification is satisfied by an MDP. This capability allows for finding some measure of the performance of the controller under models where the environment behavior can not be modeled with known probabilities, but, instead, must be modeled as non-deterministic transitions. In such cases, we still restrict the behavior of the environment according to φ_e , but do not attach probabilities to the transitions in E . Inclusion of probabilities in \bar{E} allows for the analysis of the behavior of the controller under partially unknown environments.

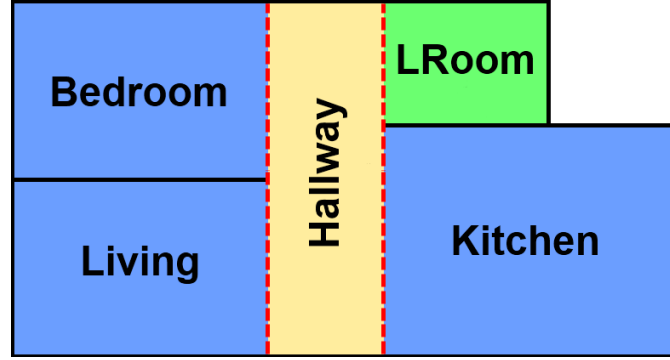


Figure 2.2: Workspace for the example of a housekeeping robot. The robot can transition across boundaries marked with dashed lines, but not those marked with solid lines.

2.5 Examples

2.5.1 Example 1: Housekeeping Robot

Scenario: A housekeeping robot moves about an environment that is divided into five separate regions, as shown in Figure 2.2. It is tasked with searching the *Bedroom*, *Kitchen*, and *Living* rooms, and performing the action *Clean* whenever it senses a *Mess*. Additionally, when the robot senses *LDone* (signaling that the laundry is done), it must return to the *LRoom* (laundry room), and perform the action *Fold*. Doing so marks the end of the robot’s task.

Controller: The controller for this example was synthesized from a set of high-level task specifications, using the LTLMoP [18] toolkit. An abridged set of the LTL specifications used to generate the controller is given below.

- $\Box(\bigcirc Mess \leftrightarrow \bigcirc Clean)$
“Clean any and all Messes”
- $\Box((\bigcirc LDone \wedge LRoom) \leftrightarrow \bigcirc Fold)$
“Fold if and only if the Laundry is Done and you are in the LRoom”
- $\Box\Diamond((\neg LDone \wedge \neg Mess) \rightarrow Bedroom)$

“If you are not sensing *Mess* or *LDone*, then visit *Bedroom*”

- $\Box\Diamond((LDone \wedge \neg Mess) \rightarrow LRoom)$

“If you are sensing *LDone* but not *Mess*, then visit *LRoom*”

Environment: The behavior of the environment propositions r_Mess and r_LDone (the prefix $r_$ is used to indicate that it is a reflection of the “real” world), is captured by the automaton shown in Figure 2.3. The transition probabilities are defined with the parameters $fMess00$, $fMess11$, and $fLDone00$, which represent the probability that r_Mess remains false, r_Mess remains true, and r_LDone remains false, respectively. Each environment proposition is also restricted according to the environmental assumptions defined in φ_e , and listed below.

- $\Box(LRoom \rightarrow \neg \bigcirc Mess)$

“No *Messes* in *LRoom*”

- $\Box(LDone \rightarrow \bigcirc LDone)$

“Once true, *LDone* stays true”

Sensors: The two sensor propositions s_Mess and s_LDone were modeled as shown in Figure 2.4. The prefix $s_$ indicates that the variables refer to the “sensed”

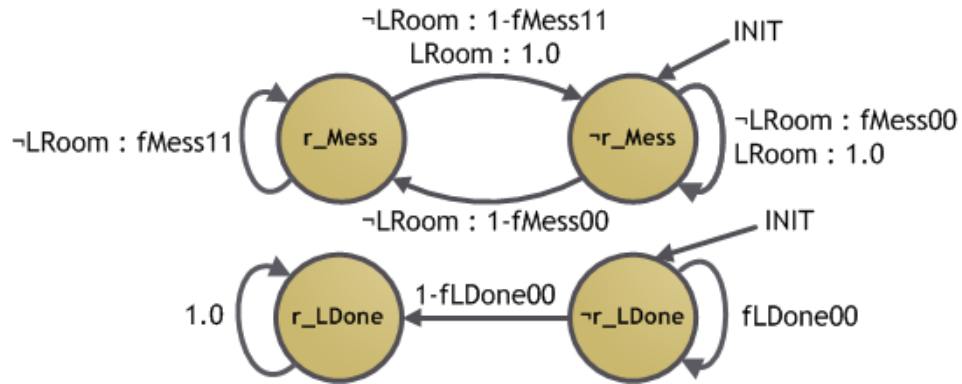


Figure 2.3: Graphical representation of the environment model, with probabilities and necessary conditions given for each transition.

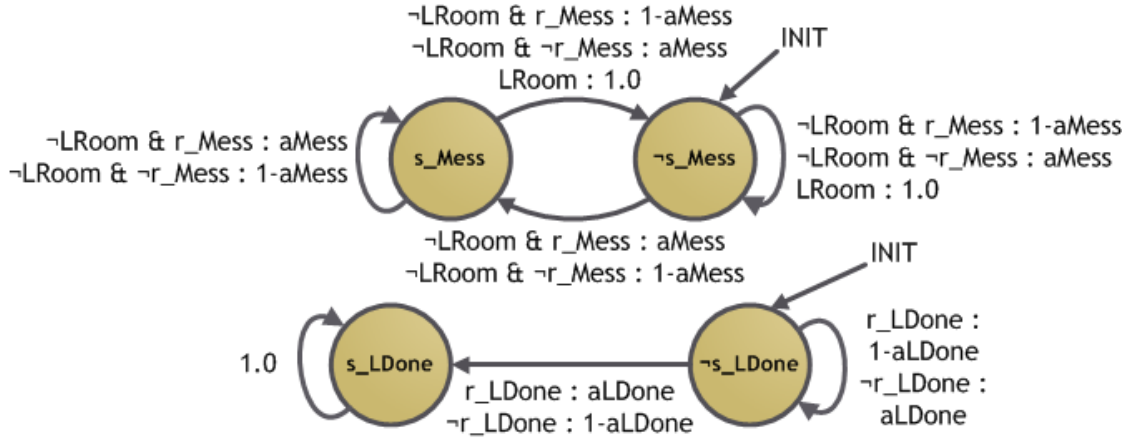


Figure 2.4: Graphical representation of the sensor model, with probabilities and necessary conditions given for each transition.

values of the corresponding environment propositions. The transition probabilities were defined by the parameters $aMess$, referring to the probability with which the s_Mess sensor correctly mimics the value of r_Mess , and $aLDone$, which represents the probability of a correct reading of r_LDone .

Analysis properties: For this example problem, the robot controller, R , consisted of 70 states. Because the task specification includes multiple liveness conditions, the robot must account for which liveness condition it is attempting to satisfy in each state, resulting in states that are not necessarily unique with respect to their labeled propositions. Composing the controller with the probabilistic environment and sensor models yielded a system with 280 states and 2346 probabilistic transitions. We analyzed the controller with respect to the following properties.

1. $\Box(Clean \leftrightarrow r_Mess)$

“Clean any and all Messes”

2. $\Box(\bigcirc Fold \leftrightarrow (\bigcirc r_LDone \wedge LRoom))$

“Fold if and only if the Laundry is Done and you are in the LRoom”

3. $\Diamond(Bedroom)$

“Visit the *Bedroom*”

While one may choose to analyze the robot behavior with respect to the specifications used for the controller synthesis, one is not necessarily restricted to those LTL formulas. The above specifications were selected to illustrate several properties of the system, and, while the first two properties are part of the specification used to synthesize the controller, the third property was not. This property is included to assess the extent to which the robot explores the workspace before ending its task.

Using the parametric model checking algorithm described earlier, we find a symbolic formula for the probability of satisfying each property, as a function of the environment event frequency and sensor accuracy. For the following results, the values of $fMess00$, $fMess11$, and $fLDone00$ were set at 0.75, 0.5, and 0.95, respectively. For the results in Figures 2.5, 2.6 and 2.8, the value of $aMess$ (accuracy of *Mess* sensor) was held at 0.75. For Figure 2.8, the value of $aLDone$ (accuracy of *LDone* sensor) was fixed at 0.95.

Using a 2.53 GHz computer with 4.00 GB of RAM, the DTMC for the complete system was composed in 0.55 seconds, while the parametric model checking for the DTMC took 3.67, 1.07, and 0.29 seconds for the three properties (given in order). Each parametric formula was then written to a function, allowing for easy evaluation for differing values of the parameters in MATLAB (evaluation of the formula for the first property requires roughly 0.04 seconds). By contrast, a single valuation of the first property is PRISM takes only 0.38 seconds; the drawback being that a full re-evaluation is required for each new set of values for the parameters.

Non-monotonic performance: Figure 2.5 shows the performance of the controller over a range of values for $aLDone$. We see that the probability of satisfying the third specification does not monotonically increase or decrease with respect to changes in the accuracy of $sLDone$. Rather, it has a minimum at a sensor accuracy of approximately 0.38. This behavior is caused by the complex interactions between the dynamics of

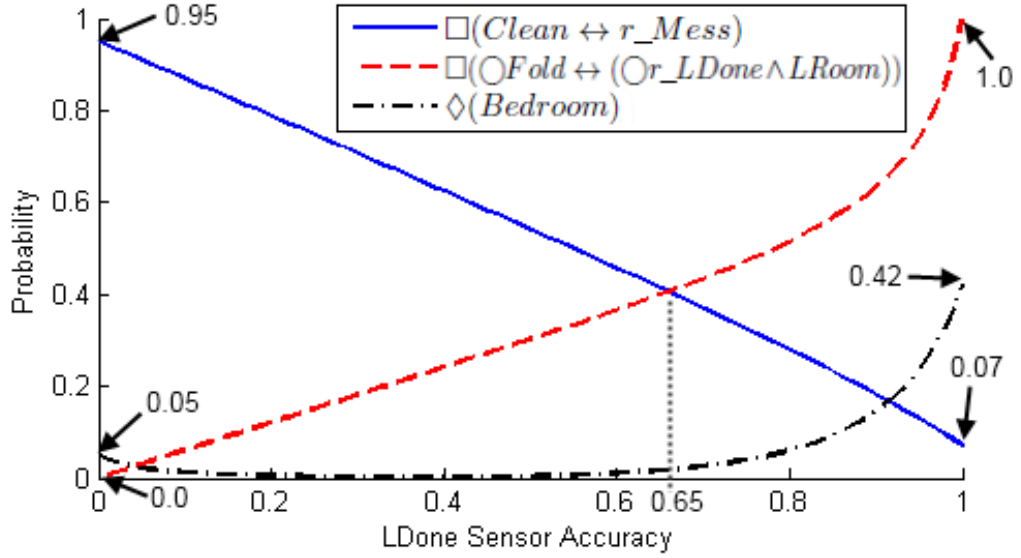


Figure 2.5: Plot of the probability with which each of the properties is satisfied for a range of *LDone* sensor accuracies, evaluated at steps of 0.01.

the system (environment frequencies and robot actions) and the sensing (sensor accuracies); the specification is satisfied when *s_LDone* remains false long enough that the robot reaches the *Bedroom*. This can occur when *r_LDone* immediately becomes true and *s_LDone* remains false, accounting for the small peak at very low sensor accuracies. Alternately, this can occur when *r_LDone* remains false and *s_LDone* correctly mimics it, accounting for the peak at high values of *a_LDone*. At intermediate accuracies, *s_LDone* is more likely to become true (either correctly or incorrectly), causing the robot to go to *LRoom* before it reaches *Bedroom*.

Sensor effect on different specifications: Figure 2.5 also shows the effects of the *LDone* sensor accuracy on the first and second specifications. We see that improvements to the accuracy of the *LDone* sensor have a negative effect on the probability with which the robot satisfies the first property, which does not depend directly on *r_LDone*. This is because, due to the low probability of *r_LDone* becoming true, a lower sensor accuracy is more likely to result in a value of true for *s_LDone*, which causes the robot to return to *LRoom*, where the value of *Mess* is fixed as false, so no errors are made. The second

specification is directly influenced by the *LDone* sensor, and improves as *s_LDone* becomes more accurate. A comparison of the two properties shows that a sensor accuracy of about 0.65 would be desired for equal performance of the controller with regards to each of the two specifications.

Bounded Analysis: Figure 2.6 shows the same analysis performed for a time bound of 15 discrete time steps, with the unbounded results shown as thinner lines, for comparison. The bounded analysis shows similar results to the unbounded analysis. The first specification shows very little change for the analyzed time-bound, since, once the *s_LDone* sensor becomes true, the specification will be satisfied, regardless of the time bound. The second specification shows improved performance in intermediate sensor accuracy ranges, owing to the possibility that the robot does not sense *s_LDone* before the end of the time bound. Finally, the probability of satisfying the third specification decreased, as the bounded time-frame reduces the chances that the robot will visit the *Bedroom* when influenced by repeated readings of *s_Mess*. One noticeable point is that the safety conditions (specifications 1 and 2) improved under the bounded execution, while the probability of satisfying the liveness condition (specification 3) decreased; this is expected due to the relationship between the two types of formulas (as discussed in Section 2.4.2), where the probability $P(\Box\varphi) = 1 - P(\Diamond\neg\varphi)$.

Picking sensor accuracies: If we wish to find the sensor accuracies needed for the robot to obtain a particular performance, it may be useful to look at the evaluation of a function over ranges of multiple sensors. Figure 2.7 shows a surface plot of the weighted average of all three specifications, with the added requirements that the first and second specifications maintain a probability greater than 0.25. The three specifications are weighted by 9, 2, and 5, respectively. This plot shows that, while the best performance results from high accuracies for both sensors, if the *Mess* sensor is inaccurate, a highly accurate *LDone* sensor may result in undesirable behavior (i.e. a point that

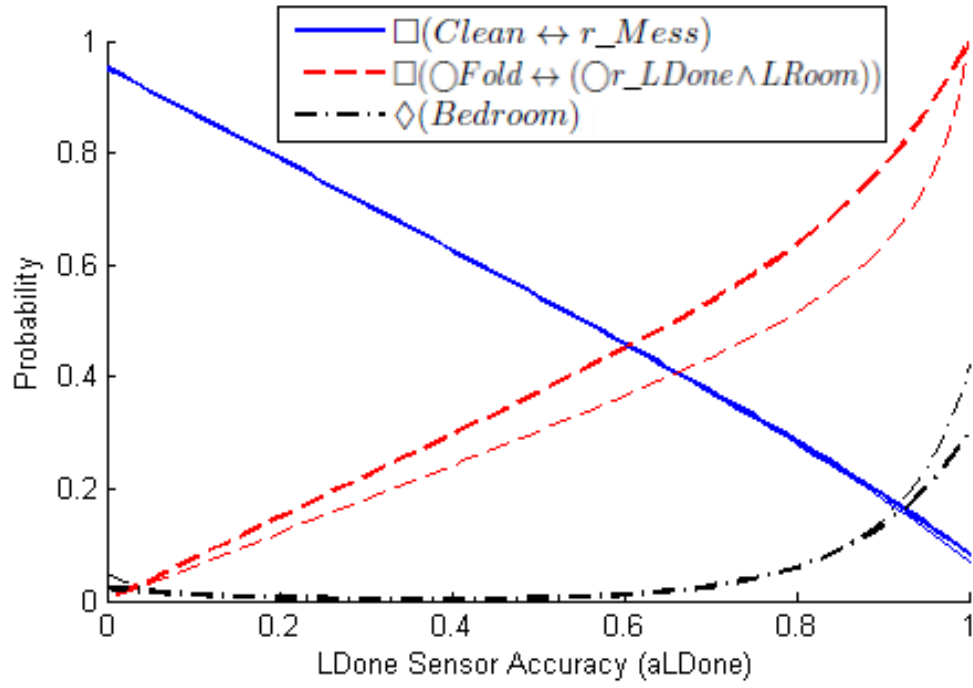


Figure 2.6: Plot of the probability with which each of the properties is satisfied for a range of $LDone$ sensor accuracies, evaluated for a time-bound of 15 discrete steps. For comparison, the unbounded results are shown as the corresponding faint lines. The probability is evaluated at steps of 0.01.

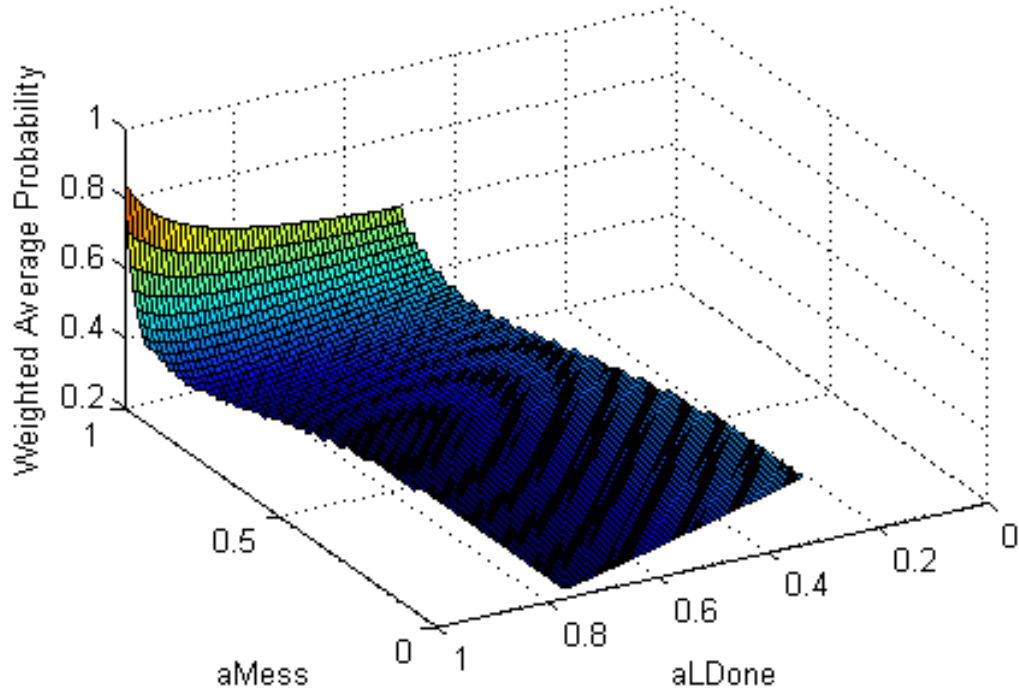


Figure 2.7: Plot of the surface representing the weighted average probability of the three properties as a function of sensor accuracies for both the *Mess* and *LDone* sensors. Additional requirements that the first and second properties have a probability greater than 0.25 are imposed, constraining the surface. The probability is evaluated at steps of 0.01.

is not part of the surface, due to the constraints).

Environment frequency effects: Figure 2.8 shows the effects caused by changing the probability with which the environment proposition r_LDone remains false (the value of $fLDone00$, in Figure 2.3). We can see from this plot that, as the probability with which r_LDone remains false increases, the first and second specifications decrease while the third increases. This is because an increase in $fLDone00$ corresponds to a lower probability that r_LDone becomes true, which, due the high accuracy of the *LDone* sensor ($a_LDone = 0.95$), creates a lower probability that the robot will return to *LRoom*. As discussed previously, a late return to *LRoom* decreases the probability with which the robot satisfies the first and second specifications, while simultaneously

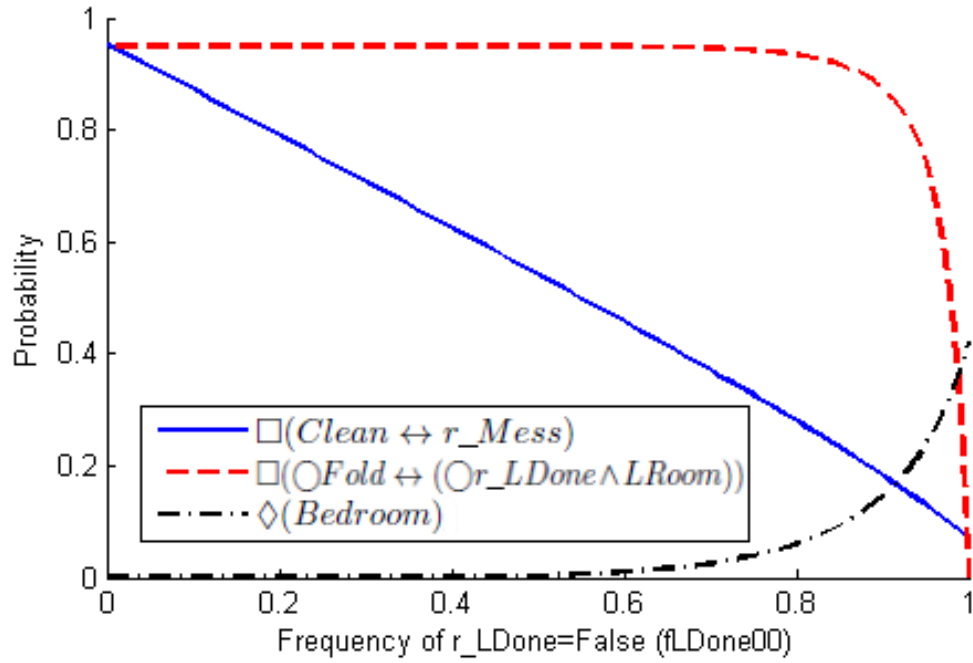


Figure 2.8: Plot of the probability with which each of the properties is satisfied for a range of probabilities with which the value of the environment proposition r_LDone remains false ($fLDone00$). The accuracy of the $LDone$ sensor is fixed at 0.95, and the probability is evaluated at steps of 0.01.

increasing the probability with which it will satisfy the third specification. The exponential decay of the probability that the second specification is satisfied can be attributed to the rapid decrease in the probability that r_LDone becomes true while, due to the fixed accuracy of the $LDone$ sensor, the probability of a false-positive reading of s_LDone remains the same. A false positive value for s_LDone when the robot returns to $LRoom$ will cause the robot to violate the specification.

2.5.2 Example 2: Taxi

Scenario: The second example we present is that of an autonomous Taxi. The robot operates in a 9-block grid of roads and intersections, with a single parking *Lot*. The discretized workspace, which has 41 different regions, is shown in Figure 2.9. The robot

I01	R01	I02	R02	I03	R03	I04
R04	Lot	R05		R06		R07
I05	R08	I06	R09	I07	R10	I08
R11		R12		R13		R14
I09	R15	I10	R16	I11	R17	I12
R18		R19		R20		R21
I13	R22	I14	R23	I15	R24	I16

Figure 2.9: Workspace for the example of a an autonomous Taxi. Roads are indicated by names beginning with *R*, while intersections are indicated by names beginning with *I*.

is tasked to continually visit each of the roads and, when it senses a *Person*, pick them up, activating the action *Passenger*. Once *Passenger* is activated, the taxi is required to take them to the *Lot* and drop them off. While doing this, the robot is required to *Stop* for any *RedLights* it senses. The final sensor the robot has is used to detect when the lot is *Full*; once it is sensed, the robot *Parks* and the task is complete.

Controller: The deterministic controller that is synthesized from the temporal logic specifications has a total of 4,102 states. Composing this controller with the models of the environment and sensors results in a probabilistic model of the system with 24,612 states. Due to the large number of states and transitions in the model, the current implementation of the parametric model checking algorithm introduced in Section 2.4 cannot efficiently calculate the probabilities with which the controller satisfies a given set of properties. As such, we use PRISM to analyze the performance of the controller. Due to the substantially larger model for this example, evaluation takes significantly longer

than for the first example; evaluating a single data point for one of the formulas required 238.54 seconds.

Environment: For this example, the environment is restricted in several ways. Firstly, a *Person* will attempt to flag-down the taxi only while in *Road* regions. In those regions, we define the appearance of a person as having a 0.2 probability of being true. The second is that only the intersections contain stop lights and, therefore, *r_RedLight* can be true only at intersections; we model it such that it has a 0.5 chance of becoming true when previously false, and a 0.75 chance of remaining true (approximating the tendency of stoplights to remain red for a short duration). The final restriction we place on the environment is that *r_Full* has a 0.1 chance of becoming true, after which it remains true.

For the situation in which the transition probabilities for the environment are unknown, we model the system as an MDP, in which the environment transitions are non-deterministic, and the sensor transitions remain probabilistic. Such a model allows us to use PRISM to analyze the minimum and maximum probabilities with which the system will satisfy the specifications.

Sensors: In each of the states where the sensor values were not restricted by the assumptions placed on the environment, the sensors were modeled such that they correctly mimicked the corresponding environment proposition with a predefined probability. For the following analysis, each of the sensors that were not being varied were held fixed at accuracies of 0.9 and 0.75 for the *s_Full* and *s_Person* sensors, respectively. The *s_RedLight* sensor accuracy was held at a value of 0.85 in all of the intersections, with the exception of intersections I02, I08, and I1. For these intersections, the sensor had a lower accuracy, held constant at 0.7, approximating intersections with poor visibility.

Analysis properties: We analyze the performance of the controller with respect to the two properties listed below.

$$1. \Box(r_RedLight \leftrightarrow Stop)$$

“*Stop* at any and all *RedLights*”

$$2. \Box(Park \rightarrow \neg Passenger)$$

“No *Passengers* when *Parking*”

Unrealizable specification: The second property, which requires that the robot never have a *Passenger* when it *Parks*, is interesting because it can not be included in the controller specification, as it is un-synthesizeable (i.e. a controller cannot be generated). We cannot guarantee that, even if the sensors are perfect, the controller will always satisfy this specification. If we analyze the specification under perfect sensor accuracies we find that the controller has a 0.625 probability of satisfying the specification. In contrast, analysis of the first property, which is part of the specification, shows 1.0 probability of satisfaction under perfect sensor accuracies.

Discontinuities in the probability: Figure 2.10 shows the results of analyzing the properties over variations in the accuracy of *s_Full*. The second specification shown in this figure has a discontinuous data point at an accuracy of 0, where the sensor will always have the wrong reading for the *Full* proposition. As a result, *s_Full* will always be true if the first *r_Full* environment value is false (once the sensor turns true it remains so, as part of the assumptions). Alternately, if *r_Full* becomes true immediately, the sensor will continually be false. For the former case, the taxi will park before picking up a passenger, guaranteeing that the specification will be satisfied. For the latter case, the taxi will never park, again guaranteeing that the specification will be satisfied. At sensor accuracies greater than 0, the latter of the two cases will no longer hold, and the sensor will eventually become true, accounting for the discontinuity.

MDP analysis: For the situation in which the environment frequencies are unknown, we can model the system as an MDP, and obtain the minimum and maximum probabilities with which our controller will satisfy the specifications. Figure 2.11 shows the

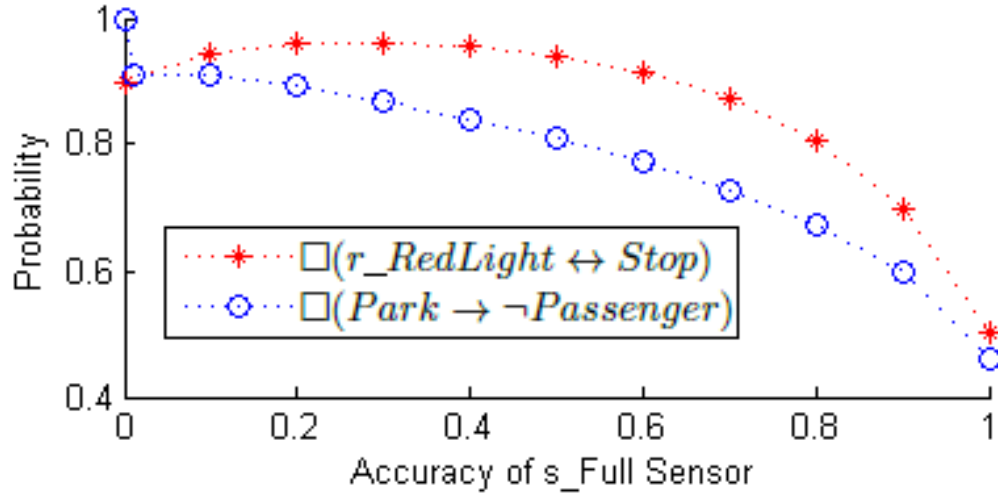


Figure 2.10: Plot of the performance of the autonomous Taxi, over a range of different sensor accuracies for the *Full* sensor.

result of such an analysis for the minimum and maximum probability with which the robot will always *Stop* if and only if it is at a *RedLight* (shown as the shaded area). We can see that, as would be expected, the probability found from the DTMC always lies within the bounds of the MDP minimum and maximum (for the specification in question). For this scenario, the resulting bounds nearly converge at a sensor accuracy of 0.5, where the frequency of r_{Full} does not affect the probability that s_{Full} becomes true (at 50% accuracy, the sensor behaves in a uniformly random fashion). The bounds do not completely converge, however, due to the influence of the *Person* and *RedLight* sensors. At s_{Full} accuracies closer to 1 or 0, the gap between the upper and lower bounds is larger, due to the influence of r_{Full} . Because we have no information about the probability with which the environment changes, the lower bound correspond to an adversarial environment, while the upper bound corresponds to a beneficial environment.

Adjusting the controller: Recognizing that the three intersections with lower *RedLight* sensor accuracy may adversely affect the performance of the robot, we can adjust the controller by adding the specification $\square \neg (\bigcirc I02 \vee \bigcirc I08 \vee \bigcirc I10)$, which requires that the robot always avoid these three regions. Analyzing this new controller under the

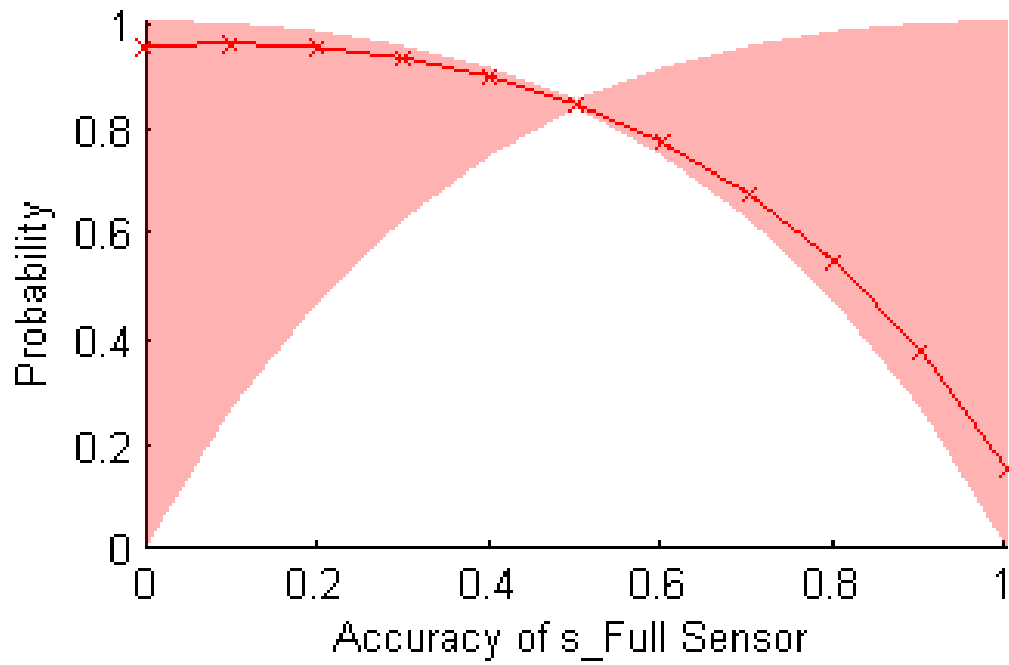


Figure 2.11: The probability of satisfying $\Box(r_RedLight \leftrightarrow Stop)$ under known environment probabilities (shown as a line with markers), as compared to the probability bound obtained by analyzing the MDP for unknown environment transition probabilities (shown by the shaded area).

Table 2.1: Table of the results for adjusting the Taxi controller to avoid intersections with poor accuracy for the *RedLight* sensor. Results are given for both specifications with the original controller and the adjusted controller, as well as the difference caused by adjusting the controller.

Bad Intersections	$\Box(r_RedLight \leftrightarrow Stop)$			$\Box(Park \rightarrow \neg Passenger)$		
	Original	Adjusted	Difference	Original	Adjusted	Difference
I02, I08, I10	0.6986	0.7004	0.0018	0.5964	0.5962	-0.0002
I01, I06, I08, I13, I15	0.4724	0.8787	0.4063	0.5335	0.4128	-0.1207

default model values (environment frequencies and sensor accuracies) yields the results shown in the second row of Table 2.1. We can see from these results that the bad intersections had negligible effect on the performance of the controller, due to the large number of intersections and the relatively small impact of the poor accuracies in just those three. If the set of bad intersections was composed of more significant regions, a similar adjustment may yield more significant changes.

The third row of Table 2.1 shows the results for the original and an adjusted controller when the intersections I01, I06, I08, I13, and I15 are the intersections with bad sensor accuracies. The results of adjusting the controller to avoid these regions shows a much more significant change. We see here that the first specification improves by more than 40%, because the bad intersections (which the controller is adjusted to avoid) are intersections that were used often by the original controller. For the second specification, the significant drop in probability is due to the round-about path the vehicle needs to take to avoid the bad intersections; taking a longer route causes the vehicle to spend more time with the passenger, increasing the probability that it has a passenger when it parks.

2.6 Conclusion

In this paper, we present a novel approach for assessing the performance of automatically synthesized controllers under the presence of erroneous sensors. By composing

a model of the system, including the behavior of the environment and sensors, we use probabilistic model checking techniques to compute the probability with which the controller will satisfy a set of high-level specifications. In the case of a partially known model of the environment or sensors, where one or more event frequencies or sensor accuracies are unknown, the model can be defined as an MDP, and upper and lower bounds can be found for the probability with which the robot will satisfy the specifications.

We discuss the calculation of parametric formulas representing the satisfaction probability as a function of the model parameters (frequency of environment changes and sensor accuracies). The algorithms presented in this paper allow for the inclusion of time bounds and the nesting of a single “next” temporal operator in the analyzed formulas. The applicability of this approach is presented with reference to two different examples.

The work presented here facilitates the assessment and redesign of a controller synthesized from temporal logic specifications and provides a foundation for further related work. In the future, we intend to extend the applicability of this approach to include actuation uncertainty in the assessment of controller performance; the primary challenge in such an extension lies in the creation of states that, due to unintended changes in the robot propositions, were not part of the synthesized controller. Such states require careful handling, especially as we consider such questions as the robot’s ability to observe its own state. Additionally, we intend to incorporate such analysis into the automated selection and synthesis of the controller, allowing for the automatic synthesis of controllers that are robust to errors in the robot’s sensors and actuators.

CHAPTER 3

**ANALYZING AND REVISING SYNTHESIZED CONTROLLERS FOR
ROBOTS WITH SENSING AND ACTUATION ERRORS**

3.1 Introduction

Recent research in the fields of robotics and automation has adapted techniques from the formal methods community to enable the synthesis of correct-by-construction robot controllers for complex tasks [8, 17, 34–36, 44, 56]. Such techniques take a specification of the desired robot behavior, and create an abstract controller that is guaranteed to satisfy those specifications (if such a controller exists). Doing so offers a number of advantages over a more traditional controller design process:

- Synthesis lowers the level of technical expertise required to design and build a controller for a complex task by abstracting away many lower-level details and avoiding traditional programming languages.
- Synthesis reduces the time investment typically required to implement a robot controller by allowing the designer to create and edit the controller directly from the task specification.
- Synthesis techniques also typically offer guarantees on the behavior of the robot with respect to the specified task; such guarantees ease the burden of validation and testing of the controller.

Research on the synthesis of robot controllers has led to a number of different approaches, each with their own advantages and disadvantages. In [16] and [61], for example, the proposed methods use feedback controllers to control the motion of the robot, while in [4] and [32] the authors sacrifice completeness and use sampling-based methods for robots with more complex dynamics. In [34] and [32] the authors use synthesized

controllers to complete complex motion tasks in static environments, while in [37] and [62], by contrast, the authors synthesize controllers that react to dynamic events in the environment.

Each of these proposed approaches provides some guarantee on the execution of the synthesized controller. The approach in [37], for example, synthesizes a controller in such a way that it is guaranteed to satisfy all of the underlying specifications, if one assumes that the abstract sensors and actuators operate without error. Such an assumption is, however, typically infeasible in real-world scenarios. As such, it is important to provide guarantees that account for the errors and uncertainties that occur when operating in complex, real-world environments.

In [34], the authors present an approach for generating a feedback control strategy from Linear Temporal Logic (LTL) specifications in such a way that all closed-loop trajectories for the controller will satisfy the specification, despite disturbances in the continuous system. In [42] and [43], the authors extend that work by modeling the uncertainty on the outcomes of the low-level motion primitives, and altering the synthesis algorithm such that it generates the motion plan that has the highest probability of satisfying the specification. In [11] the authors improve the computational feasibility of the approach via dynamic programming, and in [46] the authors present a method to maximize the probability of satisfying a specification within a specific time bound. In [10] the authors allow for time-varying observations of the robot’s environment, and generate a controller which maximizes the probability that the robot satisfies some specification (which may be reactive with respect to those observations). In [45], the authors present a Process Algebra based method for probabilistically validating robot mission software in an uncertain environment, described by random variables.

In [28] and [29], the authors assess the impact of errors in the robot’s perception of the environment. In these papers, they create sensor propositions which, in the error

free case, mimic the set of environment event propositions. By probabilistically modeling false positives and false negatives, they include the probability of erroneous sensor values in a discrete model of the overall system. They then use probabilistic model checking techniques to compute the probability that the robot satisfies some set of LTL formulas, and use the analysis to inform design choices for the synthesized controller. They apply this approach to the scenario of an autonomous car in [27].

Similarly, in [30], the authors model errors in the robot’s actuation by the inclusion of probabilistic transitions to unintended states. Again, a discrete, probabilistic model of the system is created, and a probabilistic model checker is used to compute the probability that the robot satisfies some set of LTL formulas. The paper then extends the analysis by providing a preliminary method for the automatic determination of revisions to the original specification that may result in a controller with a higher likelihood of satisfying the given task. This process of computing specification revisions for the user provides a valuable tool in the design of synthesized robot controllers, as it provides the designer with important feedback to improve the probability that the robot exhibits a particular behavior. Furthermore, by including the designer in the feedback loop (i.e. returning the suggested revision to the user, rather than automatically including it in the specification and re-synthesizing), the process helps to prevent the emergence of unexpected or unwanted behavior from the generated revisions.

The generation of revisions for controller synthesis is, at this point, largely unexplored. In a recent paper, [7] present a method for the supervised synthesis of and revision of a motion control policy for a Dubins vehicle, where the control policy is synthesized from a PCTL specification. Furthermore, the algorithms provide the user with suggestions for specification relaxation in order to improve the performance of the robot. Other related work in the literature is that of [33] and [15]. In these papers, the authors present approaches for revising an unsynthesizable temporal logic specification

by relaxing the restrictions on the initial condition of the robot in such a way that the specification becomes synthesizable. In [52], the authors describe techniques for automatically generating a concise explanation for an unsynthesizable LTL specification; the approach provides feedback to the user to aid in debugging the specification and enable the synthesis of a feasible robot controller.

The work presented in this paper builds upon the authors’ earlier work in [28], [29], and [30]. First, this paper discusses composition and analysis of a probabilistic model of the system, which includes both sensor and actuator error. The simultaneous consideration of both sensor and actuator error, which is necessary in real-world applications, requires significant adaptations of the composition algorithms presented in [29] and [30]; the new algorithms, and the required adaptations are detailed in Section 3.4. Additionally, this paper expands on the preliminary approach described in [30] for automatically generating revision suggestions for the original specification. This paper presents an improvement to the revision algorithm described in the authors’ earlier work, as well as presenting three additional methods for generating complimentary revision suggestions.

The paper is organized as follows. Section 3.2 presents some necessary background information, and Section 3.3 formally defines the problem statement. Sections 3.4 and 3.5 detail the methods for modeling and analyzing the controllers and for generating revision suggestions. The approach is illustrated by an example in Section 3.6. Concluding remarks are given in Section 3.7.

3.2 Preliminaries

3.2.1 Linear Temporal Logic

Linear Temporal Logic (LTL) is a logical formalism which allows for the expression of linear-time temporal properties. An LTL formula ϕ is defined over a set of Boolean

propositions Π . The syntax for a formula ϕ is defined recursively in Equation 3.1.

$$\phi ::= true \mid \pi \in \Pi \mid \neg\phi \mid \phi \wedge \phi \mid \bigcirc \phi \mid \phi \mathcal{U} \phi \quad (3.1)$$

The semantics of an LTL formula is defined over an infinite sequence of states σ , where a state σ_i , which occurs at time i in the sequence, is a set of truth assignments to the atomic propositions $\pi \in \Pi$. A state σ_i satisfies an LTL formula (denoted by $\sigma_i \models \phi$) as given in Equation 3.2. A sequence of states σ is said to satisfy a formula ϕ if, for all initial states σ_0 , $\sigma_0 \models \phi$.

$$\begin{aligned} \sigma_i \models \pi & \quad \text{iff } \pi \in \sigma_i \\ \sigma_i \models \neg\phi & \quad \text{iff } \sigma_i \not\models \phi \\ \sigma_i \models \phi_1 \wedge \phi_2 & \quad \text{iff } \sigma_i \models \phi_1 \text{ and } \sigma_i \models \phi_2 \\ \sigma_i \models \bigcirc\phi & \quad \text{iff } \sigma_{i+1} \models \phi \\ \sigma_i \models \phi_1 \mathcal{U} \phi_2 & \quad \text{iff } \exists k \mid k \geq i \text{ s.t. } \sigma_k \models \phi_2 \\ & \quad \text{and } \forall j \mid i \leq j < k : \sigma_j \models \phi_1 \end{aligned} \quad (3.2)$$

Intuitively, the formula $\bigcirc\phi$ is true when ϕ holds true in the next state in the sequence, and $\phi_1 \mathcal{U} \phi_2$ is true when the formula ϕ_1 holds until ϕ_2 becomes true (and ϕ_2 becomes true at some point). Additional common operators can be derived from the operators in Equation 3.1. The disjunction operator (\vee) can be defined from the negation (\neg) and conjunction (\wedge) operators, as can implication (\rightarrow) and bi-conditional (\leftrightarrow). Additionally, the temporal operators for eventually (\Diamond) and always (\Box) can be defined using the until (\mathcal{U}) operator. More details on the syntax and semantics of LTL can be found in [14].

3.2.2 Controller Synthesis

The synthesized controllers used in this work were created using the approach described in [37]. That approach requires a set of propositions \mathcal{X} that model abstract events in

the environment, which the robot must observe and react to; similarly, the state and actions of the robot are described by a set of abstract propositions \mathcal{Y} , which includes propositions to track the location of the robot in a discretized workspace.

The desired task for the robot is then described by an LTL formula that is restricted to the General Reactivity (1) class of formulas, as described in [51]. This class of formulas takes the form given in Equation 3.3.

$$\phi = (\varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e) \rightarrow (\varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s) \quad (3.3)$$

In the above formula, the specification for the robot is given as a desired robot behavior (defined by $\varphi_{i,t,g}^s$) in response to changes in the environment (defined by $\varphi_{i,t,g}^e$). The formulas φ_i^e and φ_i^s are defined as Boolean formulas B_i over $\mathcal{X} \cup \mathcal{Y}$, and represent the initial conditions of the environment and robot, respectively. The formulas φ_t^e and φ_t^s are of the form $\Box B_t$, where B_t is a Boolean formula over $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc \mathcal{X} \cup \bigcirc \mathcal{Y}$, and represent the restrictions on the possible transitions that the environment and robot can make, often referred to as *safety* specifications. Finally, the formulas φ_g^e and φ_g^s take the form $\Box \Diamond B_g$, where B_g is a Boolean formula defined over $\mathcal{X} \cup \mathcal{Y}$, and represent the goals of the environment and robot; they are referred to as *fairness* and *liveness* conditions, respectively.

From the abstracted environment \mathcal{X} and robot \mathcal{Y} propositions, as well as the task specification ϕ , a controller is synthesized using the approach described in [51]. The synthesized controller takes the form of a finite-state automaton $\mathcal{A} = \{S, S_0, \mathcal{X}, \mathcal{Y}, \delta, L\}$, where S is the set of states, and $S_0 \subseteq S$ is the set of possible initial states. \mathcal{X} and \mathcal{Y} are the environment and robot propositions, as described above. $\delta : S \times 2^{\mathcal{X}} \rightarrow S$ is a transition function which maps a state and set of environment (input) propositions to a successor state, and $L : S \rightarrow 2^{\mathcal{Y}}$ is a labeling function which maps each state to a subset of the robot propositions \mathcal{Y} . As additional information, the synthesis algorithm can provide a function $X_0 : S_0 \rightarrow 2^{\mathcal{X}}$ which maps initial states to sets of environment

propositions, and a ranking function $\Gamma : S \rightarrow \mathbb{N}$ which maps each state to the index of the liveness condition that state is working towards completing.

3.2.3 Probabilistic Model Checking

A number of different options exist for performing probabilistic model checking, depending on the type of model and the type of properties to be checked. The work in this paper relies on model checking a Discrete-Time Markov Chain (DTMC) model against one or more LTL properties. The probabilistic model is defined as a DTMC $\mathcal{D} = \{Q, Q_0, \Delta, \Pi, \mathcal{L}, \Gamma\}$, where Q is the set of states and $Q_0 \subseteq Q$ is the set of initial states. The transition function $\Delta : Q \times p \rightarrow Q$ maps one state to another, with probability p . The labeling function $\mathcal{L} : Q \rightarrow 2^\Pi$ maps each state to a subset of the propositions Π , and the ranking function $\Gamma : Q \rightarrow \mathbb{N}$ maps each state to the index of its corresponding liveness condition.

While a number of different algorithms and off-the-shelf model checkers exist, the work presented in this paper uses PRISM probabilistic model checker [38], which offers, among other capabilities, the ability to compute the probability that a DTMC satisfies a particular LTL formula, and is available online.

3.3 Problem Statement

The work presented in this paper considers correct-by-construction controllers such that, when the robot's sensors and actuators operate without error, the controller is guaranteed to satisfy all of its underlying specifications. This paper investigates the affects of errors in sensing and actuation on the behavior of the robot, and the revision of the controller to improve the behavior of the robot under those erroneous conditions. To include these effects, the behavior of the environment, sensors, and actuation of the robot are all be

defined probabilistically.

The changes in the environment are described by a set of probabilities $P(X' | X, Y)$ that represent the changes in the abstract environment propositions \mathcal{X} . In this set of probabilities, each new environment configuration $X' \subseteq \mathcal{X}$ (that is, the subset of environment propositions that are true in the next step) is modeled to depend only on the previous configuration of the environment $X \subseteq \mathcal{X}$ (the subset of environment propositions that are true in the current step) and the previous configuration of the environment $Y \subseteq \mathcal{Y}$ (the subset of robot propositions that are true in the current step).

To model the sensors, an additional set of propositions $\overline{\mathcal{X}}$ is used to represent the robot's perception of the environment propositions such that $\overline{\mathcal{X}} = \{\overline{x}_i | x_i \in \mathcal{X}\}$. The evolution of the sensor propositions is given by the set of probabilities $P(\overline{X}' | X', \overline{X}, Y)$, such that the new sensor configuration \overline{X}' is dependent only on the new environment configuration X' and the current sensor and robot configurations, \overline{X} and Y respectively. In this model, the sensor error is represented through false positives (where $\overline{x}_i' = true$ and $x_i' = false$) and false negatives (where $\overline{x}_i' = false$ and $x_i' = true$) in how the sensor propositions mimic the environment propositions.

To model the actuation error, changes in the robot configuration are modeled by the set of probabilities $P(Y' | \overline{Y}', Y)$, where the new robot configuration Y' is dependent only on the previous robot configuration Y and the intended next robot configuration \overline{Y}' . This model accounts for actuation error through differences between the intended next robot configuration \overline{Y}' (as determined by the synthesized controller) and the actual configuration Y' that results from attempting to make that transition.

Each of these sets of probabilities (for the environment, sensors, and robot actuation) can be estimated using statistical data gathered from experiments or simulations. In [43], for example, the authors use experimental data to determine the probabilistic outcomes of different motion primitives during the operation of their robot. Similarly, in [27] the

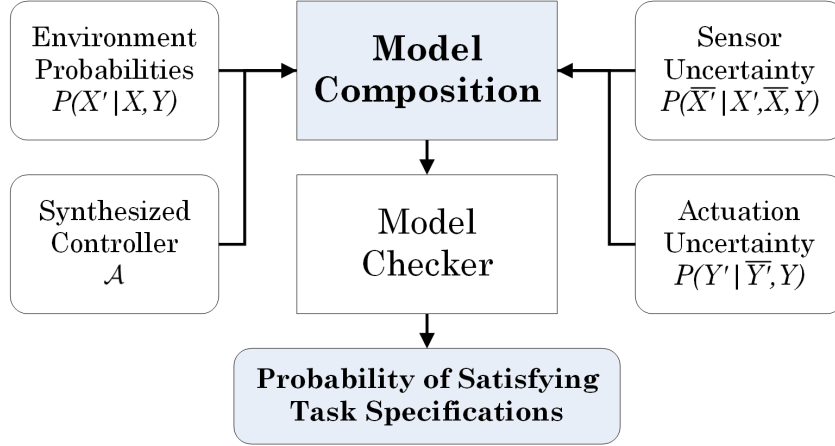


Figure 3.1: Diagram of the approach used to compose and analyze a probabilistic model of the robot, including the effects of sensor and actuator error

authors use statistical information gathered from a simulation to model the changes in the abstracted environment and sensors for an autonomous vehicle.

Problem: Given a synthesized robot controller \mathcal{A} and probabilistic models of the environment $P(X' | X, Y)$, imperfect sensors $P(\bar{X}' | X', \bar{X}, Y)$, and imperfect robot actuation $P(Y' | \bar{Y}', Y)$, find the probability that a robot using the synthesized controller will satisfy a set of LTL specifications. Furthermore, provide revision suggestions to the user to improve the probability that the robot satisfies the specifications.

3.4 Modeling and Analysis

An overview of the approach presented in this paper for composing and analyzing a model of the robot with sensor and actuator error is shown in Figure 3.1. The model of the probabilistic system is composed from the synthesized controller and probabilistic models of the environment, sensors, and actuation, it is then analyzed with a probabilistic model checker to calculate the likelihood that the robot will satisfy a given set of task specifications.

3.4.1 Sensor Error

Algorithm 6 describes the process of composing the environment probabilities and sensor error with the synthesized controller to create a probabilistic model of the system with imperfect sensors. This algorithm is a modified version of the one given in [29], and has been changed to facilitate the integration of actuator error by including the correspondence between each probabilistic state and its originating automaton state. Additionally, the algorithm was changed such that it creates new probabilistic states for any transition with probability greater than zero, and includes transitions to deadlock states and states which correspond to unconnected automaton states. This change allows for arbitrary probabilistic models of the sensors (of the form $P(\overline{X}'|X', \overline{X}, Y)$), instead of requiring that the sensor model adhere to any environmental assumptions, as was the case in [29].

Algorithm 6 takes as input the synthesized automaton \mathcal{A} , the set of initial environment configurations X_0 , the controller's ranking function Γ , the set of sensor propositions $\overline{\mathcal{X}}$, and probabilistic models of the environment and sensors, $P(X'|X, Y)$ and $P(\overline{X}'|X', \overline{X}, Y)$, respectively. The output of this algorithm is the probabilistic model \mathcal{D} of the system with sensor error, and a function $\Lambda : Q \rightarrow S$ which maps each state in the probabilistic model to its originating state in the synthesized automaton. The resulting probabilistic model \mathcal{D} includes transitions to incorrect states in the automaton as well as to states that were not in the synthesized controller, due to false positives and false negatives in the robot's sensor propositions.

Consider a scenario where a robot is required to pass from one room to another through a doorway which may be either closed or open. The synthesized automaton, then, would include a state where the robot moves through the open door, and one where the robot waits at the closed door. In including the sensor error, Algorithm 6 would model not only if the door was open but also if the robot sensed that the door was

open. The algorithm would create new states that include both the sensor values and the state of the environment, and add transitions based on the specified probabilities. As a result, the model would include transitions to a state where the robot attempts to drive through a closed door (which it erroneously senses is open) and to a state where the robot mistakenly waits for the open door (which it senses is closed) to open, as well as to the correct states where it drives through the open door or waits for the closed door to open.

Intuitively, the algorithm loops through each state in the automaton (line 3), and extracts the set of propositions for the sensors, which correspond to the incoming transitions (lines 4-8). The algorithm then adds a new state to the probabilistic system, labeled with the environment-sensor-robot configuration, and with the appropriate rank and automaton mappings (lines 9-12). As each state is created, it is added to the set Q^* of states for which the outgoing transitions are undefined (line 11). If the corresponding automaton state was an initial state, it is also added to the set of initial states for the probabilistic system (line 6).

While there are states in the set Q^* , the algorithm selects one ($q_i \in Q^*$) and removes it from the set (lines 13-14). It then extracts the appropriate labels for the environment, sensors, and robot, at that state (line 15). Lines 16 and 17 then loop through each allowable configuration of the environment and sensors, and the probability of transitioning to those configurations is calculated in Line 18.

If there exists some state s_k in the synthesized automaton such that there is a transition from s_i to s_k that is labeled with the given sensor values \overline{X}_k , the robot configuration is extracted from that state, and a transition is created to the appropriate state in the probabilistic system (lines 19-20). If a state q_{jk} with the appropriate labels exists, and that state corresponds to the automaton state s_k , a transition is created to it (lines 21-22); if no such state already exists, a new state is created and added to the set of states missing

Algorithm 6 Include sensor error in the system model.

```

1: procedure SENSEERR( $\mathcal{A} = \{S, S_0, \mathcal{X}, \mathcal{Y}, \delta, L\}, X_0, \Gamma, \overline{\mathcal{X}}, P(X'|X, Y), P(\overline{X}'|X', \overline{X}, Y)$ )
2:    $Q = \emptyset, Q_0 = \emptyset, Q^* = \emptyset, \Delta = \emptyset, \Lambda = \emptyset, \Gamma_q = \emptyset$ 
3:   for  $s_i \in S$  do
4:     if  $s_i \in S_0$  then
5:        $X_i = X_0(s_i)$ 
6:        $Q_0 = Q_0 \cup q_i$ 
7:     else
8:        $X_i \subseteq \mathcal{X}$  s.t.  $\exists s_h \in S \mid (s_h, X_i, s_i) \in \delta$ 
9:        $\overline{X}_i = X_i, Y_i = L(s_i)$ 
10:       $\mathcal{L}(q_i) = X_i \cup \overline{X}_i \cup Y_i$ 
11:       $Q = Q \cup q_i, Q^* = Q^* \cup q_i$ 
12:       $\Lambda(q_i) = s_i, \Gamma_q(q_i) = \Gamma(s_i)$ 
13:   while  $Q^* \neq \emptyset$  do
14:     Choose  $q_i \in Q^*, Q^* = Q^* \setminus q_i$ 
15:      $s_i = \Lambda(q_i), X_i = \mathcal{L}(q_i) \cap \mathcal{X}, \overline{X}_i = \mathcal{L}(q_i) \cap \overline{\mathcal{X}}, Y_i = \mathcal{L}(q_i) \cap \mathcal{Y}$ 
16:     for  $X_j \in 2^{\mathcal{X}}$  s.t.  $p(X_j|X_h, Y_i) > 0$  do
17:       for  $\overline{X}_k \in 2^{\overline{\mathcal{X}}}$  s.t.  $p(\overline{X}_k|X_j, \overline{X}_i, Y_i) > 0$  do
18:          $p_{ijk} = p(X_j|X_i, Y_i) \cdot p(\overline{X}_k|X_j, \overline{X}_i, Y_i)$ 
19:         if  $\exists s_k \in S$  s.t.  $(s_i, \overline{X}_k, s_k) \in \delta$  then
20:            $Y_k = L(s_k)$ 
21:           if  $\exists q_{jk} \in Q$  s.t.  $\mathcal{L}(q_{jk}) = X_j \cup \overline{X}_k \cup Y_k$  and  $\Lambda(q_{jk}) = s_k$  then
22:              $\Delta = \Delta \cup (q_i, p_{ijk}, q_{jk})$ 
23:           else
24:              $\mathcal{L}(q_{jk}) = X_j \cup \overline{X}_k \cup Y_k$ 
25:              $Q = Q \cup q_{jk}, Q^* = Q^* \cup q_{jk}$ 
26:              $\Lambda(q_{jk}) = s_k, \Gamma_q(q_{jk}) = \Gamma(s_k)$ 
27:              $\Delta = \Delta \cup (q_i, p_{ijk}, q_{jk})$ 
28:         else if  $\exists s_h, s_k \in S$  s.t.  $L(s_h) = L(s_i)$  and  $(s_h, \overline{X}_k, s_k) \in \delta$  then
29:            $(s_h^*, s_k^*) = \operatorname{argmin}_{(s_h, s_k)} ((\Gamma(s_i) - \Gamma(s_h)) \bmod \max_{s \in S} (\Gamma(s)))$ 
30:           if  $\exists q_{jk} \in Q$  s.t.  $\mathcal{L}(q_{jk}) = X_j \cup \overline{X}_k \cup Y_k$  and  $\Lambda(q_{jk}) = s_k^*$  then
31:              $\Delta = \Delta \cup (q_i, p_{ijk}, q_{jk})$ 
32:           else
33:              $\mathcal{L}(q_{jk}) = X_j \cup \overline{X}_k \cup Y_k$ 
34:              $Q = Q \cup q_{jk}, Q^* = Q^* \cup q_{jk}$ 
35:              $\Lambda(q_{jk}) = s_k^*, \Gamma_q(q_{jk}) = \Gamma(s_k^*)$ 
36:              $\Delta = \Delta \cup (q_i, p_{ijk}, q_{jk})$ 
37:         else
38:            $Y_k = Y_i$ 
39:            $\mathcal{L}(q_{jk}) = X_j \cup \overline{X}_k \cup Y_k$ 
40:            $Q = Q \cup q_{jk}$ 
41:            $\Lambda(q_{jk}) = \text{None}, \Gamma_q(q_{jk}) = \Gamma_q(q_i)$ 
42:            $\Delta = \Delta \cup (q_i, p_{ijk}, q_{jk}) \cup (q_{jk}, 1, q_{jk})$ 
43:   return  $\mathcal{D} = \{Q, Q_0, \Delta, \Pi = \mathcal{X} \cup \overline{\mathcal{X}} \cup \mathcal{Y}, \mathcal{L}, \Gamma_q\}, \Lambda$ 

```

outgoing transitions Q^* , and the appropriate transition is created in Δ (lines 23-27)

If there is no appropriate transition in the synthesized automaton (out of s_i), then the algorithm looks for an identically labeled transition out of another state which shares the same configuration as s_i (line 28). If any such states exist, the algorithm takes the one with the most closely preceding goal rank (line 29), and creates the appropriate transition. Again, if the destination state is already in the probabilistic system, the algorithm adds a transition to it (lines 30-31); if no such state exists, a new one is created and added to the set Q^* (lines 32-36).

Finally, if there are no transitions in the synthesized automaton which match the considered labels, a sink state (with no outgoing transitions) is created, and a transition is added to that state (lines 37-42).

3.4.2 Actuator Error

Algorithm 7 describes a similar process, which introduces the actuation error into the probabilistic model. This algorithm is adapted from the one presented in [30], and is slightly modified to incorporate the actuation error into an existing DTMC, rather than creating a probabilistic model from the synthesized automaton. These modifications enable the algorithm to account for sensor configurations (in addition to environment and robot configurations), and to include the probabilities that were computed for each transition in Algorithm 6.

The inputs to the algorithm are the DTMC \mathcal{D} of the probabilistic model with sensor error (from Algorithm 6), the sets of environment \mathcal{X} , sensor $\overline{\mathcal{X}}$, and robot \mathcal{Y} propositions, the function Λ which maps states in \mathcal{D} to states in \mathcal{A} , and the probabilistic model of the actuation error $P(Y'|\overline{Y}', Y)$. The outputs of the algorithm are the adjusted probabilistic model \mathcal{D} , which includes actuation error in addition to sensor error, and the expanded automaton-state mapping function Λ . The new probabilistic model includes

new states and new transitions that are a result of unintended changes in the state of the robot.

Consider, again, the scenario where a robot is required to pass through a doorway, when open. The probabilistic model that results from Algorithm 7 would now include the possibility of the robot attempting and failing to exit through the door when it senses that it is open. This involves the creation of new states, in which the robot attempts and fails to move through a doorway it senses to be open. If such states are not planned for in the automaton, they result in deadlock states in the probabilistic model (no outgoing transitions).

Algorithm 7 Include actuation error in the model.

```

1: procedure ACTERR( $\mathcal{D} = \{Q, Q_0, \Delta, \Pi, \mathcal{L}, \Gamma_q\}, \mathcal{X}, \overline{\mathcal{X}}, \mathcal{Y}, \Lambda, P(Y'|\overline{Y'}, Y)$ )
2:    $\Delta_{Act} = \emptyset$ 
3:   for  $(q_i, p_{ij}, q_j) \in \Delta$  do
4:      $X_i = \mathcal{L}(q_i) \cap \mathcal{X}, \overline{X}_i = \mathcal{L}(q_i) \cap \overline{\mathcal{X}}, Y_i = \mathcal{L}(q_i) \cap \mathcal{Y}$ 
5:      $X_j = \mathcal{L}(q_j) \cap \mathcal{X}, \overline{X}_j = \mathcal{L}(q_j) \cap \overline{\mathcal{X}}, \overline{Y}_j = \mathcal{L}(q_j) \cap \mathcal{Y}$ 
6:     for  $Y_k \in 2^{\mathcal{Y}}$  s.t.  $p(Y_k|\overline{Y}_j, Y_i) > 0$  do
7:       if  $Y_k = \overline{Y}_j$  then
8:          $\Delta_{Act} = \Delta_{Act} \cup (q_i, p_{ij} \cdot p(Y_k|\overline{Y}_j, Y_i), q_j)$ 
9:       else if  $\exists(q_h, q_k) \in Q \times Q$  s.t.  $(q_h, p_{hk}, q_k) \in \Delta$ 
           and  $\mathcal{L}(q_k) = X_j \cup \overline{X}_j \cup Y_k$  then
10:          $q_l = \operatorname{argmin}_{q_k} ((\Gamma_q(q_j) - \Gamma_q(q_k)) \bmod \max_{q \in Q} (\Gamma_q(q)))$ 
11:          $\Delta_{Act} = \Delta_{Act} \cup (q_i, p_{hl} \cdot p(Y_k|\overline{Y}_j, Y_i), q_l)$ 
12:       else
13:          $\mathcal{L}(q_{jk}) = X_j \cup \overline{X}_j \cup Y_k$ 
14:          $Q = Q \cup q_{jk}$ 
15:          $\Lambda(q_{jk}) = \Lambda(q_j)$ 
16:          $\Delta_{Act} = \Delta_{Act} \cup (q_i, p_{ij} \cdot p(Y_k|\overline{Y}_j, Y_i), q_{jk})$ 
17:          $\Delta_{Act} = \Delta_{Act} \cup (q_{jk}, 1, q_{jk})$ 
18:   return  $\mathcal{D} = \{Q, Q_0, \Delta_{Act}, \Pi, \mathcal{L}, \Gamma\}, \Lambda$ 

```

This algorithm proceeds by looping through each transition in the input DTMC (line 3) and extracting the propositions associated with each state in the transition (lines 4-5). It then loops through each robot configuration that is a possible outcome of the attempted transition (line 6) and, if that outcome is the intended one, it adjusts the probability of

the transition according to the actuation error model, and stores the transition in the new transition function Δ_{Act} (lines 7-8). Otherwise, if the outcome is not the intended one, but matches another state in the system, a new transition is created to the matching state with the most closely preceding goal (lines 9-11). Finally, if no such state exists in the system, a new one is created and added to the system (lines 12-16); additionally, this new state is set to be a sink state, which transitions to itself with probability 1 (line 17).

3.4.3 Model Analysis

The model \mathcal{D} that is the output of Algorithm 7 is a discrete-time model which describes the probabilistic behavior of the robot which is controlled by the synthesized automaton \mathcal{A} , and operating with imperfect sensors and actuators. This model can be analyzed with respect to various LTL formulas as described in [28], [29], and [30]. These formulas may be any valid LTL formula describing the behavior of the robot, and are not restricted to the original task specification. Typically, these formulas are used to compute the probability that the robot exhibits a particular behavior in response to changes in the robot's environment.

As in the previous papers, PRISM probabilistic model checker [38] was used to find the probabilities used for the analysis presented in Section 3.6. It was also used to perform the model-checking functionality in Algorithms 8-11. The PRISM model checking software is available online at <http://www.prismmodelchecker.org/>.

3.5 Revision Suggestion

An overview of the presented approach for generating revision suggestions is shown in Figure 3.2. After composing a probabilistic model of the system, as described in Section 3.4, the model is analyzed with a probabilistic model checker, and used to provide

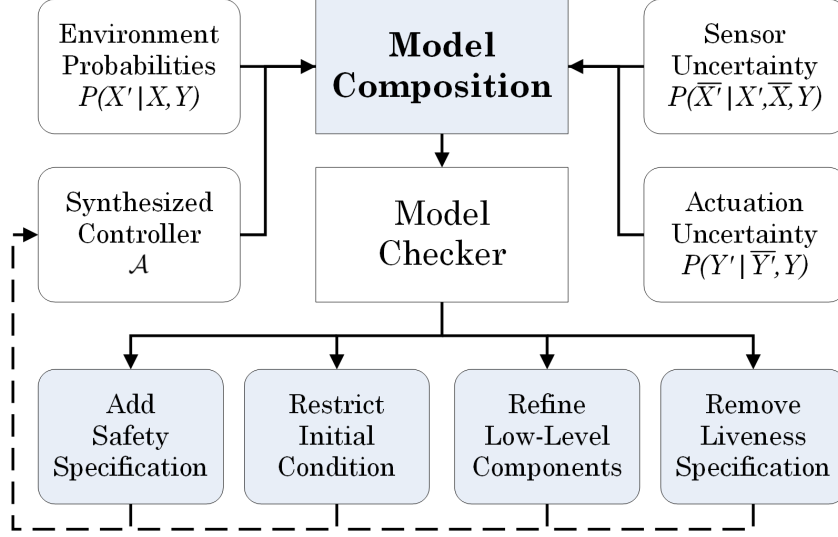


Figure 3.2: Diagram of the presented approach for providing feedback to the user in the form of an additional safety specification, a restriction on the initial condition, a low-level component to refine, or a liveness specification to remove.

feedback to the user in the form of an additional safety specification, a restriction on the initial condition, a low-level component to refine, or a liveness specification to remove.

3.5.1 Safety Specification Revisions

Algorithm 8 describes the process used to provide specification revisions in the form of additional safety formulas (φ_t^s) to be added to the original task specification. The inputs to the algorithm are the probabilistic system \mathcal{D} , a formula ϕ , and a maximum number of steps to check N_{max} . The formula ϕ is a Boolean formula over $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc \mathcal{X} \cup \bigcirc \mathcal{Y}$, which represents a desired behavior for the robot (typically, this takes the form of a conjunction of safety formulas, less the \square operator). N_{max} determines the maximum number of steps ahead of each state the algorithm will check for violations of the specification ϕ ; the value of N_{max} serves as a termination criterion for the for the algorithm in the case that it is unable to find a suitable revision formula. The output of this algorithm is a set of safety formulas Ψ which may be added to the original specification to modify the

robot behavior.

Returning to the example scenario of a robot attempting to move through an open door, consider a situation in which there are two doorways that lead out of the room. If one of those doorways opens and closes more frequently, Algorithm 8 may detect that attempting to move through that doorway is the most likely source of collisions with a closed door, and return a revision suggestion that requires the robot to always avoid using that particular doorway.

Algorithm 8 Suggest additional safety requirements.

```

1: procedure SAFETYREVS( $\mathcal{D} = \{Q, Q_0, \Delta, \mathcal{X} \cup \overline{\mathcal{X}} \cup \mathcal{Y}, \mathcal{L}, \Gamma\}, \phi, N_{max}$ )
2:    $Q_{goal} = \{q \in Q : \Gamma(q) \neq \Gamma(q') \ \forall q' \text{ s.t. } (q, \cdot, q') \in \Delta\}$ 
3:    $Q_{deadlock} = \{q \in Q : (q, 1, q) \in \Delta\}$ 
4:    $Q_{check} = Q \setminus (Q_0 \cup Q_{goal} \cup Q_{deadlock})$ 
5:   for  $N = 0 : N_{max}$  do
6:      $Q^* = \emptyset$ 
7:     for  $q_i \in Q_{check}$  do
8:        $D_i = \{Q, q_i, \Delta, \mathcal{X} \cup \overline{\mathcal{X}} \cup \mathcal{Y}, \mathcal{L}, \Gamma\}$ 
9:        $\varphi = \neg\phi \wedge t = N$ 
10:       $p \leftarrow \text{MODELCHECK}(\mathcal{D}_i, \varphi)$ 
11:      if  $p > 0$  then
12:         $Q^* = Q^* \cup (q_i, p)$ 
13:       $Y'_T = Y'_F = \mathcal{Y}; X'_T = X'_F = \mathcal{X}; \Psi = \emptyset$ 
14:      while  $Q^* \neq \emptyset$  and  $Y'_T \cup Y'_F \neq \emptyset$  do
15:         $(q_j, p) = \text{argmax}_{(q_i, p) \in Q^*}(p)$ 
16:         $Y'_T = Y'_T \cap \mathcal{L}(q_j); Y'_F = Y'_F \setminus \mathcal{L}(q_j)$ 
17:         $X'_T = X'_T \cap \mathcal{L}(q_j); X'_F = X'_F \setminus \mathcal{L}(q_j)$ 
18:        if  $Y'_T \cup Y'_F \neq \emptyset$  then
19:           $\psi = \Box((\bigwedge_{x \in X'_T} \bigcirc x \wedge \bigwedge_{x \in X'_F} \neg \bigcirc x)$ 
20:             $\rightarrow \neg(\bigwedge_{y \in Y'_T} \bigcirc y \wedge \bigwedge_{y \in Y'_F} \neg \bigcirc y))$ 
21:          if  $\psi \notin \Psi$  then
22:             $\Psi = \Psi \cup \{\psi\}$ 
23:             $Q^* = Q^* \setminus (q_j, p)$ 
24:          if  $\Psi \neq \emptyset$  then
25:            return  $\Psi$ 
26:      return  $\emptyset$ 

```

The algorithm proceeds, first, by computing the set of goal states (where the robot satisfies a liveness condition and begins working towards the next one, line 2), and the

set of deadlock states (where the robot has no transitions out of that state, line 3). The algorithm then reduces the set of states to check Q_{check} (line 4) to only those states that are not initial states (which are considered in Algorithm 9), goal states (which are considered in Algorithm 11), or deadlock states (which are either sink states that are not in the synthesized automaton, or are goal states).

Next, beginning with a value of $N = 1$, and increasing incrementally, the algorithm loops through each state in the system (lines 5-7). It then sets the initial state of the model to be the current state, and checks for the probability that the model violates ϕ in exactly N steps (lines 8-10). If the calculated probability is non-zero, it adds the state/probability pair to the list Q^* .

After this is completed for each state in the system, the algorithm initializes a set of propositions to track which robot and environment propositions are true or false, in an incrementally increasing set of states (line 13). It then loops through the state/probability pairs in Q^* , and chooses the one with the highest probability of violating ϕ in N steps (lines 14-15). In lines 16-17, the algorithm reduces each set of propositions to only those propositions which match the value in the current state (i.e. a proposition is removed from X'_T or Y'_T if it is false in the current state, and removed from X'_F or Y'_F if it is true in the current state). The resulting lists contain exactly the set of propositions that have a consistent value (true or false) over every state that is pulled from the list Q^* .

After each set is reduced for the current state, if the set of robot propositions (either true or false) are non-empty, a formula $\psi = \Box(\bigcirc X' \rightarrow \neg \bigcirc Y')$ is created (lines 18-19), such that if the environment propositions take the values stored in the lists X'_T and X'_F , the robot is required to avoid the configuration stored in the lists Y'_T and Y'_F . If this equation ψ is not already in the list of safety revisions Ψ , then it is added (lines 20-21). The current state/probability pair is then removed from the list Q^* (line 22), and the algorithm continues the while-loop.

After each value of N is checked, if one or more revisions has been found, the algorithm returns the set of LTL formulas Ψ (lines 23-24). If no revisions have been found for the current value of N , the algorithm increments the value and repeats the process. If no results are found for any value $0 \leq N \leq N_{max}$, the algorithm terminates with no revisions (line 25).

The set Ψ that is output by Algorithm 8 is a set of LTL safety formulas that may be added to the original specification, to force the synthesized controller to avoid a certain behavior that is exhibited by the set of state-pairs used to create the formula. The first formula added to the set is the most specific (it corresponds to avoiding a single transition in the original automaton) and each subsequent formula becomes progressively more general (corresponding to avoiding progressively larger sets of transitions).

3.5.2 Initial-State Revisions

Another type of automated feedback is described in Algorithm 9 which provides specification revisions in the form of restrictions on the initial configuration of the robot (φ_i^s). The inputs to this algorithm are the probabilistic model of the system \mathcal{D} and the LTL formula ϕ describing the desired behavior of the robot. Unlike in Algorithm 8, this formula is not restricted beyond being an LTL formula over the set of propositions $\mathcal{X} \cup \mathcal{Y}$. The output of Algorithm 9 is a Boolean formula ψ over the propositions in \mathcal{Y} , which describes the best initial configuration of the robot, from the set of possible initial states.

Algorithm 9 proceeds by looping through each initial state $q_0 \in Q_0$ and restricting the probabilistic system to only that initial state (lines 3-4). The algorithm then uses the probabilistic model checker to find the probability that this restricted system satisfies the desired behavior ϕ (line 5). This probability, along with the robot labels on the initial state, are stored in the set P (lines 6-7). After this process is completed for each initial state, the algorithm finds the stored pair with the highest average probability over all

Algorithm 9 Suggest a restriction of the initial state.

```

1: procedure INITREVS( $\mathcal{D} = \{Q, Q_0, \Delta, \mathcal{X} \cup \bar{\mathcal{X}} \cup \mathcal{Y}, \mathcal{L}, \Gamma\}, \phi$ )
2:    $P = \emptyset$ 
3:   for  $q_0 \in Q_0$  do
4:      $\mathcal{D}_0 = \{Q, q_0, \Delta, \mathcal{X} \cup \mathcal{Y}, \mathcal{L}, \Gamma\}$ 
5:      $p \leftarrow \text{MODELCHECK}(\mathcal{D}_0, \phi)$ 
6:      $l = \mathcal{L}(q_0) \cap \mathcal{Y}$ 
7:      $P = P \cup (p, l)$ 
8:    $(p^*, l^*) = \text{argmax}_{(p, l) \in P} [\text{mean}_{(p', l') \in P | l' = l}(p')]$ 
9:    $\psi = \bigwedge_{y \in l^*} y \wedge \bigwedge_{y \in \mathcal{Y} \setminus l^*} \neg y$ 
10:  return  $\psi$ 

```

elements in the set which share the same labels (line 8). These labels are then used to create a Boolean formula ψ over the set of propositions \mathcal{Y} which restricts the initial robot configuration to a single set of values (line 9).

The formula ψ which is created by Algorithm 9 can be added to the set of specifications for the robot's initial configuration φ_i^s to improve the overall behavior of the robot (with respect to the input formula ϕ), by restricting the initial state of the robot to the single configuration that has the highest probability of satisfying ϕ . It should be noted that this algorithm does not attempt to restrict the initial configuration of the environment, which is assumed to be outside of the user's control. Furthermore, if the set of initial states Q_0 does not contain multiple states, with different robot configurations, including ψ in the synthesis specifications will have no effect (as the initial configuration is already restricted to that valuation).

3.5.3 Low-Level Component Revisions

The third type of automated revision presented in this paper is that of suggesting revisions to a low-level component that is represented by one of the abstract propositions that are used to represent the robot's sensing or actuation. Algorithm 10 describes the process used to perform a sensitivity analysis on the low-level components, and iden-

tify the component $(p(\cdot) \in P(\cdot) \subseteq P(\overline{X'} | X', \overline{X}, Y) \cup P(Y' | \overline{Y'}, Y))$ for which the robot is most sensitive to improvements in performance. The inputs to the algorithm are the probabilistic system \mathcal{D} , the desired behavior ϕ (which, like in Algorithm 9, can be any LTL formula over the propositions in $\mathcal{X} \cup \mathcal{Y}$), and the set of low-level component probabilities to be analyzed $P(\cdot)$. The output of the algorithm is the single low-level component $p(\cdot) \in P(\cdot)$ which, if improved, would have the greatest improvement in the behavior of the robot (with respect to ϕ).

Algorithm 10 Suggest revisions to the low-level components (abstract actuators and sensors).

```

1: procedure HWREVS( $\mathcal{D} = \{Q, Q_0, \Delta, \mathcal{X} \cup \overline{\mathcal{X}} \cup \mathcal{Y}, \mathcal{L}, \Gamma\}, \phi, P(\cdot)$ )
2:    $p \leftarrow \text{MODELCHECK}(\mathcal{D}, \phi)$ 
3:    $P_\delta = \emptyset$ 
4:   for  $p(\cdot) \in P(\cdot)$  do
5:      $\Delta_\delta \leftarrow \text{replace } p(\cdot) \text{ with } p(\cdot) + \delta \text{ for all } p(\cdot) \in \Delta$ 
6:      $\mathcal{D}_\delta = \{Q, Q_0, \Delta_\delta, \mathcal{X} \cup \mathcal{Y}, \mathcal{L}, \Gamma\}$ 
7:      $p_\delta \leftarrow \text{MODELCHECK}(\mathcal{D}_\delta, \phi)$ 
8:      $P_\delta = P_\delta \cup (p_\delta, p(\cdot))$ 
9:    $(p_\delta^*, p(\cdot)^*) = \text{argmax}_{(p_\delta, p(\cdot)) \in P_\delta} (p_\delta - p)$ 
10:  return  $p(\cdot)^*$ 

```

Algorithm 10 first finds the probability that the nominal model satisfies the input formula ϕ (line 2). It then loops through each low-level component $p(\cdot) \in P(\cdot)$ and modifies the transition function in the original model by replacing each occurrence of $p(\cdot)$ with a slightly higher value $p(\cdot) + \delta$, and adjust the other probabilities in Δ accordingly (lines 4-5). The system model is then adjusted for the new transition function, and checked to find the probability that it satisfies the desired behavior ϕ (lines 6-7). The probability, along with the component that was adjusted, is then stored in the set P_δ (line 8). Finally, the algorithm finds the probability/component pair with the largest increase in probability (over the nominal), and returns that component to the user (lines 9-10).

The component probability that is returned by Algorithm 10 represents the single actuation or sensing component that, if improved, would have the largest positive impact

on the behavior of the robot (with respect to ϕ). This component could be improved separately by the user, by making changes to the underlying abstraction or algorithms, or by investing in more accurate hardware.

3.5.4 Liveness Condition Revisions

The final automated revision discussed in this paper is the identification of a specific liveness condition from the set of liveness formulas (φ_g^s), in pursuit of which the robot is most likely to exhibit an undesirable behavior. Algorithm 11 describes this process, where the inputs to the function are the probabilistic system \mathcal{D} , and the desired behavior ϕ . As in Algorithm 8, the desired behavior ϕ is restricted to a Boolean formula over $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc\mathcal{X} \cup \bigcirc\mathcal{Y}$. The output of this algorithm is the index γ^* of the liveness condition which is most likely to result in a violation of the desired behavior ϕ .

Algorithm 11 Suggest removal of liveness condition.

```

1: procedure LIVENESSREVS( $\mathcal{D} = \{Q, Q_0, \Delta, \mathcal{X} \cup \overline{\mathcal{X}} \cup \mathcal{Y}, \mathcal{L}, \Gamma\}, \phi$ )
2:    $P_g = \text{zeros}(\Gamma_{max})$ 
3:   for  $q_j \in Q$  do
4:      $\varphi = \phi \mathcal{U} (!\phi \wedge \bigcirc q = q_j)$ 
5:      $p \leftarrow \text{MODELCHECK}(\mathcal{D}, \varphi)$ 
6:      $P_g[\Gamma(q_j)] = P_g[\Gamma(q_j)] + p$ 
7:    $\gamma^* = \text{argmax}_{\gamma \in 1:|P_g|} (P_g[\gamma])$ 
8:   return  $\gamma^*$ 

```

This algorithm begins by creating a set of probabilities (initialized to 0) to track the probability that the robot violates ϕ while pursuing each particular goal (line 2). It then loops through each state q_j in the system, and finds the probability that the first time the robot violates ϕ is while transitioning to that state (lines 3-5). This value is then added to stored value that corresponds to the goal being pursued in state q_j (line 6). The algorithm then returns the goal index γ^* that has the largest stored value for the probability of causing the initial violation of ϕ (lines 7-8). Note that, because the

formula φ that is being analyzed finds the probability that each state is the *first* to violate ϕ , the sum of all entries in P_g will be no greater than 1.

3.5.5 Notes on Revision Suggestions

Algorithms 8-11 describe four complimentary methods for providing feedback to the user in the form of suggested changes to the task specification or low-level components. It should be noted, however, that the suggested safety revisions (Algorithm 8) and the suggested liveness revisions (Algorithm 11) are not guaranteed to result in an improvement of the behavior of the robot. In each case, the suggested revision would alter the behavior of the robot by prohibiting the current most problematic behavior or by removing a behavior that leads to violation of the specifications. Synthesizing a new controller with the suggested revision will change the behavior of the robot, and may be probabilistically worse than the behavior being avoided by the revision. Additionally, the addition of a new safety specification may even cause the overall specification to become unsynthesizable.

It is also worth noting that the removal of a liveness condition represents a more significant change in the original specification than the introduction of a safety specification (Algorithm 8) or a restriction of the initial condition (Algorithm 9). In this case, the revision causes a change in the task goals, and so must be handled with care. As such, it is typically better to revise the controller as described in Sections 3.5.1-3.5.3 prior to removing a liveness condition from the original task specification. The dramatic impact of such a change also underlines the value of keeping the user involved in the revision process, as an automation of this process would converge to a specification without any liveness conditions, where the robot could easily satisfy the specifications without needing to act.

Finally, it should be noted that one can easily automate the process of adding or

removing specification revisions and synthesizing a new controller, which can be used to create a new probabilistic model; the probabilistic performance of the new model can then be assessed in comparison to the original model, to determine if the revision results in an improvement in performance. This can then be used to automatically compare and prioritize the different types of revisions and provide the user with only those which provide a significant improvement in the behavior of the robot. The authors have, in fact, implemented such functionality, though it is not discussed in detail in this paper.

In terms of computation, the most expensive part of each of these algorithms is that of model checking a probabilistic model to find the probability that it satisfies an LTL formula. Such LTL model checking is, in general, exponential in the size of the formula and polynomial in the size of the given model [9]. As such, Algorithms 8 and 11 are typically the most time consuming to run, as they usually require the most calls to the “ModelCheck” function. Compared to model checking, the other calculations in each of the algorithms have negligible computational costs.

3.6 Examples

Scenario: In this example, the robot is tasked with continuously visiting three different regions (labeled G_1 , G_2 , and G_3 in Figure 3.3), while avoiding an adversarial robot that patrols the five regions in the middle of the map (R_1 - R_5 in the figure). The robot can (imperfectly) sense the location of the adversarial robot, and (imperfectly) move between adjacent regions in the map.

A subset of the LTL formulas used to synthesize the controller are given below, where A_{R_i} is an environment proposition describing the location of the adversary. The first formula specifies that the adversary can only transition between adjacent regions, while the second formula requires that the robot avoid being in the same region as the adversary. The final equation specifies the goals for the robot: repeatedly visiting re-

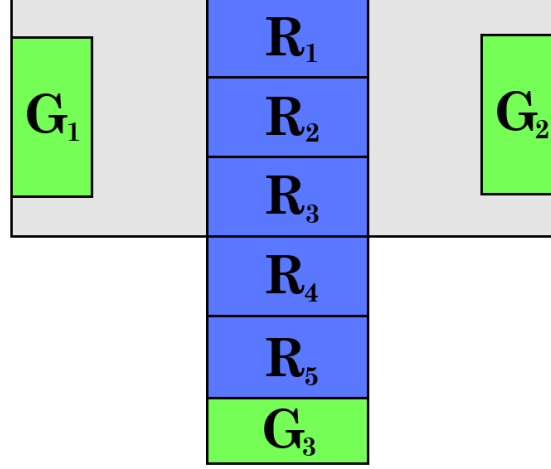


Figure 3.3: Discretized map of the workspace for the example problem.

gions G_1 , G_2 , and G_3 .

- $\Box \bigwedge_{i \in \{1:5\}} (A_{R_i} \rightarrow \bigvee_{j \in \{i-1:i+1\}} \bigcirc A_{R_j})$
- $\Box \bigwedge_{i \in \{1:5\}} (\bigcirc A_{R_i} \rightarrow \neg \bigcirc R_i)$
- $\Box \Diamond G_1 \wedge \Box \Diamond G_2 \wedge \Box \Diamond G_3$

The synthesized controller has 75 states, with the initial state of the robot being restricted to G_1 , G_2 , or G_3 . When moving from G_1 to G_2 , the robot traverses through region R_1 when the adversary is in regions R_2 - R_5 and through region R_2 when the adversary is in region R_1 . When moving to and from G_3 , the robot waits until the adversary traverses to R_1 or R_2 before entering regions R_3 - R_5 . If the sensing of the adversary's location and the movement of the robot both execute without failure, the synthesized controller is guaranteed to satisfy the specified behavior; if the robot operates imperfectly, however, it may end up in the same region as the adversary, violating the specification.

Probabilities: In this example, the movement of the adversarial is modeled such that it as a 0.25 probability of staying in its current region on any step, and a 0.75 probability of moving to one of the adjacent regions (split evenly among the adjacent regions). The

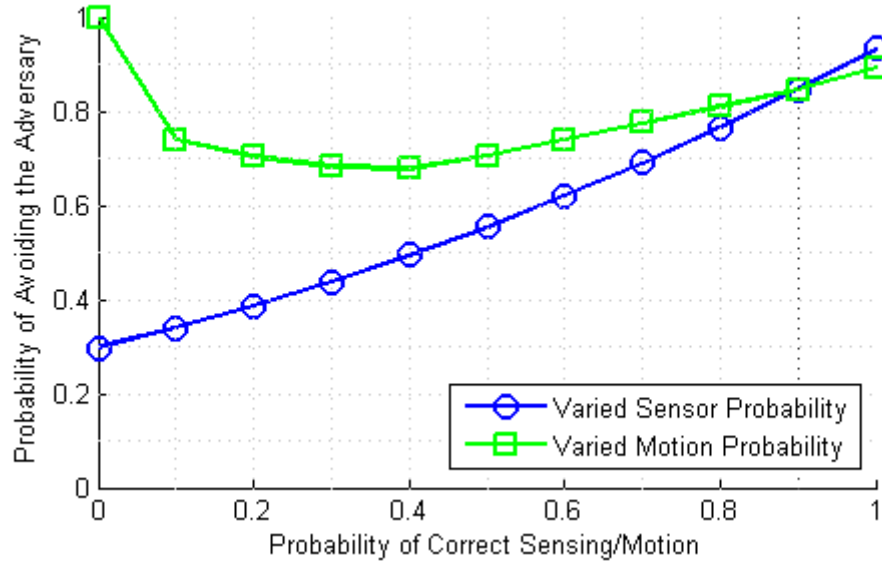


Figure 3.4: Analysis results for the adversarial robot example: probability that the robot avoids the adversary.

nominal model of the robot's sensors and actuators are such that the robot has a 0.9 probability of correctly determining the location of the adversary (with the remaining 0.1 being split between the regions adjacent to the actual location of the adversary) and a 0.9 probability of moving to the intended next region (with a 0.1 probability of erroneously remaining in it's current region).

Analysis: Figure 3.4 shows the resulting probability that the robot will avoid the region with the adversary, over a time bound of 25 discrete transitions. The probability that the robot correctly sensed the location of the adversary and the probability that the robot moved to its intended next region were each independently varied between 0 and 1, and in each case the other probability was held constant at the nominal probability of 0.9. The model was analyzed to find the probability that the robot satisfied the LTL formula given below.

$$\Box(\bigwedge_{i \in \{1:5\}} \neg(\bigcirc A_{R_i} \wedge \bigcirc R_i))$$

These results show that, as the probability that the robot correctly senses the location

of the adversary increases, the robot will be more likely to avoid the unwanted contact with the adversary. Because of the other error present in the model (the motion of the robot), even when the sensor is perfect (i.e. it has a probability of 1.0 of correctly detecting the location of the adversary) the robot has a less than perfect probability of avoiding the adversary over the time bound (0.933).

When analyzed over a range of values for the probability that the robot correctly moves when directed to do so by the automaton, the resulting probabilities show that when the robot can never move correctly it will always avoid the adversary. This can be attributed to the fact that the robot will be completely unable to move and will simply remain in region G_1 where it cannot encounter the adversary, but it will be unable to satisfy its goals. As the likelihood that the robot moves correctly increases (until about 0.4), this probability decreases, before rising again to a value of 0.891 when the robot moves without error. This behavior is due to the additional time required for the robot to reach a dangerous region when the probability of correctly moving is low; if the robot does not move into the central regions (where the adversary patrols), it will not violate the analyzed formula.

By contrast, Figure 3.5 shows the probability that the robot satisfies all 3 of the goals at least once, within the given time bound. In this case, the model was analyzed to find the probability that the robot satisfied the following formula.

$$\Diamond G_1 \wedge \Diamond G_2 \wedge \Diamond G_3$$

This figure shows that, due to the likelihood that the robot remains stopped for long portions of time, low probabilities on the motion of the robot result in low likelihoods of satisfying all 3 goals. By contrast, errors in the sensing of the location of the adversary have relatively little effect on the likelihood that the robot satisfies each of its goals at least once; in this case, the sensor error prevents the robot from satisfying its goals only when causing it to enter deadlock states, for which the synthesized controller has no

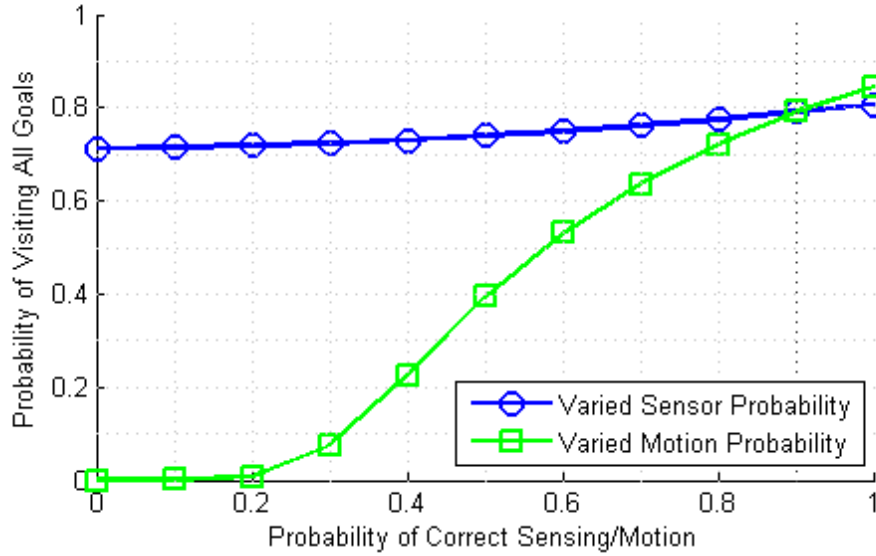


Figure 3.5: Analysis results for the adversarial robot example: probability that the robot satisfies all 3 goals within the time bound.

prescribed transition.

Revisions: In addition to the above analysis, Algorithms 8-11 were used to find revisions to the task specification and lower-level components for this example. In each case, the analysis focused on the safety specifications for the robot, requiring that it avoid being in the same region as the adversary.

To find safety revisions for the controller specification, the specified undesirable behavior was given by the formula $\phi = \bigwedge_{i \in \{1:5\}} (A_{R_i} \wedge R_i)$, describing the situation in which the robot enters the same region as the adversary. Algorithm 8 was then run with a maximum step value of $N_{max} = 5$, which was sufficient to find a safety revision before terminating. The resulting safety revision found by the algorithm $(\Box(\bigcirc \neg A_{R_1} \wedge \bigcirc \neg A_{R_2} \wedge \bigcirc \neg A_{R_4}) \rightarrow \bigcirc \neg R_4)$, when included in the original specification, requires that the robot avoid region R_4 while the adversary is in R_3 or R_5 . The new controller results in an increase of 0.10 (at the nominal model probabilities) in the probability that the robot always satisfies the formula ϕ within the time bound. Figure 3.6 shows the probability that the robot satisfies $\Box(\bigwedge_{i \in \{1:5\}} \neg(\bigcirc A_{R_i} \wedge \bigcirc R_i))$ for the original (solid

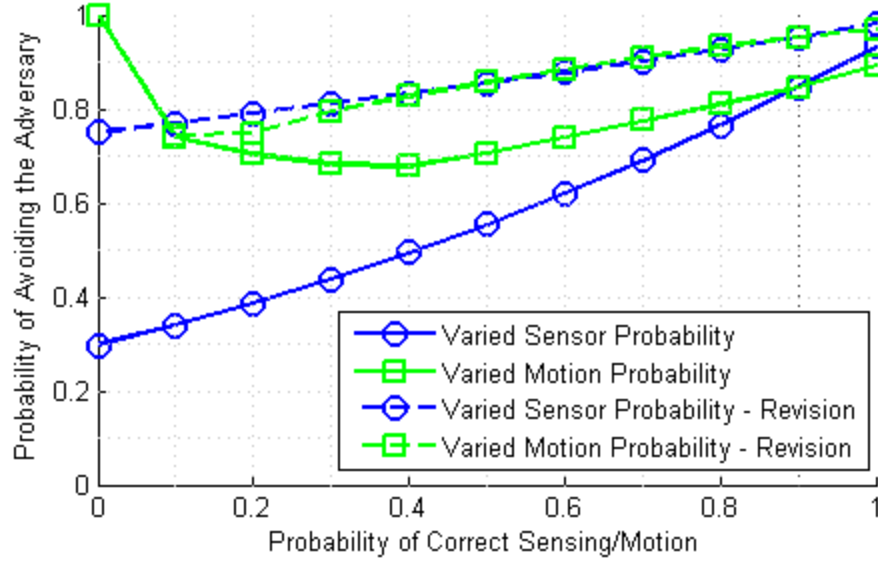


Figure 3.6: Analysis results for the revised controller in the adversarial robot example: probability that the robot avoids the adversary.

lines) and revised (dashed lines) controllers. The results shown in this figure show an improvement in the behavior of the robot for all model probabilities close to the nominal values.

An initial state revision was found (as in Algorithm 9) for the LTL formula $\phi = \Box \bigwedge_{i \in \{1:5\}} \neg(A_{R_i} \wedge R_i)$, requiring that the robot never end up in the same region as the adversary. The resulting revision to the initial condition $\psi = G_3$ specifies that the robot is most likely to succeed over the analyzed time bound (25 discrete transitions) when the robot starts in region G_3 . At the nominal model probabilities, Algorithm 9 found that the likelihood of satisfying ϕ from an initial state in G_3 was 0.8697, while the robot had probabilities of 0.8225 and 0.8160 of satisfying ϕ when beginning in regions G_2 and G_1 , respectively. This result is due to the process of traveling to region G_3 being more likely to cause a failure than the process of leaving G_3 ; as such, by beginning in G_3 , the robot is more likely to avoid that entry corridor within the analyzed time-bound.

Algorithm 10 was used to perform a sensitivity analysis on the low level components for sensing the location of the adversary, and the motion of the robot. As before,

the formula $\phi = \Box \bigwedge_{i \in \{1:5\}} \neg(A_{R_i} \wedge R_i)$ was tested against, requiring that the robot never end up in the same region as the adversary. The resulting analysis revealed that improvements to the robot's sensing would have a greater positive impact on the overall performance of the robot than would improvements to the robot's motion. This is unsurprising, given the results shown in Figure 3.4, which show a greater upward slope (at the nominal point) for the line representing changes in sensor accuracy.

Finally, Algorithm 11 was used to find the specified goal in pursuit of which the robot was most likely to violate $\phi = \bigwedge_{i \in \{1:5\}} (A_{R_i} \wedge R_i)$ by finding itself in the same region as the adversary. The result of Algorithm 11 found that the robot was most likely to violate ϕ while pursuing its third goal $\Box \Diamond G_3$ (with a probability of 0.4998), than while pursuing either of its first two goals (probabilities of 0.2505 and 0.2164 for $\Box \Diamond G_1$ and $\Box \Diamond G_2$, respectively). Given the layout of the workspace (shown in Figure 3.3), this is unsurprising, as the robot must traverse through several dangerous regions when attempting to move to region G_3 . Note that, because the analysis is performed for a bounded time-frame, there is a positive probability that the robot will successfully avoid the adversary for the entire time bound, and the sum of the probabilities of failure for the goals is less than 1.

Each of the revision methods presented in Section 3.5 gives, for this example, a revision suggestion that results in an improvement in the behavior of the robot. In fact, multiple revisions could be included simultaneously to have a greater impact on the behavior of the robot. In doing so, it is important to consider the effects of each revision. For example, if one were to remove the third liveness condition they would likely not want to restrict the initial condition of the robot to G_3 , as the removal of the liveness condition removes the need to use that dangerous corridor at all, unless the robot were to start in region G_3 .

3.7 Conclusion

This paper presents a method for accounting for errors in the sensing and actuation of mobile robots, when controlled by a correct-by-construction automaton that is synthesized from a set of high-level task specifications. The paper discusses the composition of a Discrete-Time Markov Chain model of the system, which includes the probabilistic errors in sensing and actuation, and the analysis of that model with respect to a set of LTL formulas.

Furthermore, four complementary methods are given for providing feedback to the user in a semi-automated fashion. By adding safety specifications, restricting the initial state of the robot, improving the performance of a particular low-level component, or removing a particularly troublesome liveness condition, the user may be able to improve the overall performance of the robot when operating with the modeled error.

The presented approach seeks to advance the field of synthesized robot controllers by relaxing the standard assumption of perfect sensing and actuation. Future work by the authors will focus on improving the revision process, and on incorporating the system uncertainty in the synthesis process to yield a correct-by-construction controller that is more robust to errors in sensing and actuation.

CHAPTER 4

AUTOMATIC SYNTHESIS OF HIGH-LEVEL CONTROLLERS FOR ROBOTS WITH IMPERFECT SENSING AND ACTUATION

4.1 Introduction

Recent work in robotics has developed techniques that enable the automatic synthesis of abstract robot controllers from high-level task specifications [4, 8, 17, 32, 34, 35, 56, 62]. Such techniques offer several advantages over more traditional programming, such as more intuitive task description, a lower technical barrier to entry, and formal guarantees on the behavior of the resulting controller. The approach presented in [37], for example, synthesizes a controller in a manner which guarantees that the resulting controller will satisfy all of its underlying specifications. This guarantee, however, is predicated on the assumption that the robot operates with perfect sensing and actuation.

The work presented in this paper relaxes that assumption, and describes a method for accounting for the probabilistic behavior of the robot (as caused by erroneous sensors and actuators) when synthesizing the controller. The resulting controller not only satisfies the specifications assuming perfect sensing and actuation, but minimizes the probability that a robot operating with the modeled errors in its sensing and actuation will violate a prescribed behavior. The approach allows the user to specify the robot's task at an abstract level, and is general enough to be applied to a wide array of robotics problems. Additionally, by adjusting the formula that is used in the synthesis process and describes the desired behavior, the resulting controller can be quickly and easily modified to more likely avoid or exhibit a specified behavior. Coupled with the work described in [28–30], the probabilistic behavior of the synthesized controller can be quickly analyzed with respect to the modeled error probabilities to determine the probabilistic behavior of the robot and provide feedback to the user in the form of possible

revisions to the task specification.

4.1.1 Related Work

In [42] and [43], the authors present an approach for synthesizing complex motion plans that maximize the probability that the robot satisfies a temporal logic specification when the outcomes of the robot's motion is uncertain (due to errors in localization and motion). In [20], the authors present an approach for synthesizing a control policy from a temporal logic specification for use in an initially unknown, stochastic environment; the policy is then iteratively updated to create an approximately optimal controller for a learned model. The approach presented in this paper differs from these works in that it is used to synthesize a controller which reacts to observed changes in the robot's environment; this enables the synthesis of controllers for a wider range of tasks in which the robot changes its behavior based on observed changes in its environment.

In [46], the authors present a method for synthesizing a control strategy which maximizes the probability that a robot will satisfy a given set of temporal logic specifications within a specific time-frame. They consider the case where the robot operates in a time-varying environment, for which the environment changes can be modeled stochastically. The approach presented in this paper differs from this work by the inclusion of abstract actions for the robot, in addition to the motion problem. The inclusion of abstract, non-motion actions allows for the specification of more complicated tasks, in which the robot may manipulate or influence its environment.

In [10], the authors describe a method for synthesizing a controller which maximizes the probability of satisfying a specification while reacting to time-varying observations of the robot's environment. These observations are formulated as propositions that may be true or false (with some probability) at each vertex in the partitioned environment; the truth value of each proposition can be determined by the robot only once it reaches

the vertex in question. In [57], the authors present a method for synthesizing a controller which maximizes the probability that a deterministic plant satisfies a temporal logic specification, when interacting with other, probabilistic agents. Both of these approaches allow for the synthesis of controllers that react to changes in the robot’s environment, but use different formulations for the environment than the abstract event formulation used in this work. The formulation used in this work is identical to that used in [28–30], allowing those techniques (the analysis and revision of synthesized controllers) to be directly applied, as well as providing a formulation that is general enough to be applied to a wide range of applications.

Other related work includes approaches for synthesizing controllers which are optimal with respect to some other continuous metric. The authors in [54] describe a method for automatically generating an optimal robot trajectory which satisfies robot tasks specified in temporal logic formulas and minimizes the time between reaching states satisfying a specified subformula. In [59], the authors present an approach for synthesizing an optimal robot controller to minimize the weighted average transition cost with respect to a given transition system, while satisfying a set of temporal logic constraints. The authors of [60] present a method for generating an optimal controller for discrete-time dynamical systems, from a robot task specification given as a set of logical formulas. These formulas are converted into mixed-integer linear constraints to reduce computation effort.

Of particular interest here is the work presented in [26], which presents an approach for automatically synthesizing an optimal robot controller from temporal logic specifications in adversarial environment. The optimal controller minimizes a cost function that accounts for the required robot’s action cost and the possible delay due to changes in the environment. This approach allows for the synthesis of a controller that is *reactive*; that is, the robot changes its behavior based on observed changes in the state of

the environment. The work presented in this paper uses the method described in [26], and presents a cost function to synthesize a controller that minimizes the probability that the robot violates a set of specifications, while operating with sensing and actuation uncertainty. Specifically, this work considers a different, but related, problem to that in [26]: rather than creating a cost function to penalize time delays and travel distance for a task, this paper presents a cost function that penalizes behaviors with high probabilities of failure.

This work compliments the authors previous work in [28–30], which describe methods for modeling errors in the robots sensing and actuation, and including those errors in a composed model of the system. This model can then be analyzed to determine the probability that the robot, when operating with the synthesized controller, will satisfy a particular temporal logic formula. This analysis can also be leveraged to automatically provide the user with suggested revisions to the original controller specification, in order to improve the probabilistic behavior of the robot. The probabilistic models used for the environment behavior, and the sensor and actuator error, are used in this paper to directly synthesize a controller which minimizes the probability of violating a particular temporal logic formula.

4.1.2 Example 1

As an illustrating example, consider a robot that operates in the workspace shown in Figure 4.1. The robot is tasked with exploring each of the four goal regions ($G_1 - G_4$, shown in green), in search of a *target*. Once a *target* is found, the robot must *mark* it and return to the *Base* region. During execution, the robot must avoid the *Door* region when it is *closed*, and must avoid getting *stuck* (which may occur, with varying likelihoods, when moving through the non-goal regions).

The remainder of this paper is structured as follows. Section 4.2 describes neces-

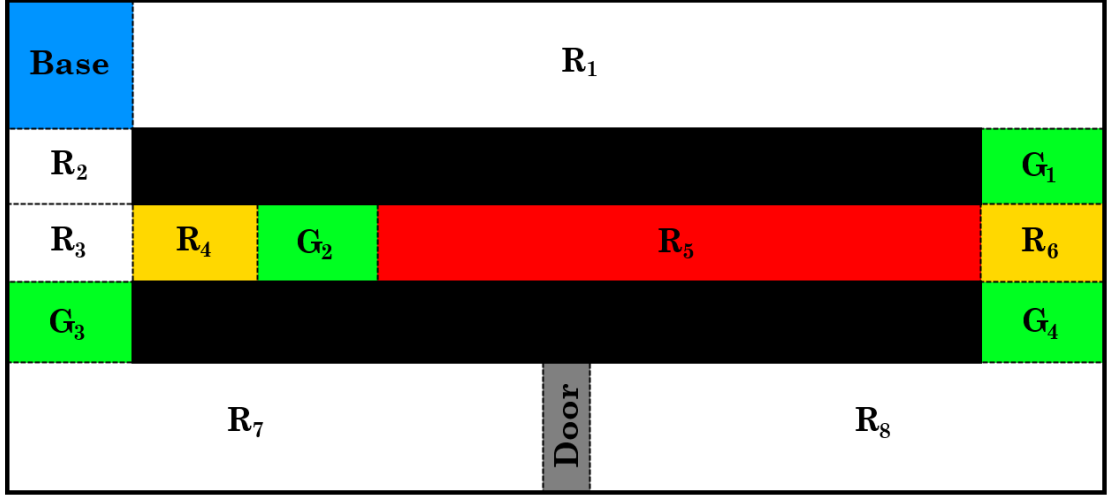


Figure 4.1: Discretized workspace for Example 1.

sary background information. Section 4.3 defines the problem that is being addressed. Section 4.4 then describes the cost function used to synthesize the controller that minimizes the probability that the robot violates a given temporal logic formula. Section 4.5 presents and compares several controllers created using the described cost function and minimizing different temporal logic formulas. Finally, some concluding remarks are given in Section 4.6.

4.2 Preliminaries

4.2.1 Linear Temporal Logic

The work in this paper uses Linear Temporal Logic (LTL) formulas to express temporal properties of the system. The syntax for an LTL formula ϕ , over a set of Boolean propositions Π , is defined recursively in (4.1).

$$\phi ::= true \mid \pi \in \Pi \mid \neg\phi \mid \phi \wedge \phi \mid \bigcirc \phi \mid \phi \mathcal{U} \phi \quad (4.1)$$

The semantics of an LTL formula ϕ is defined over an infinite sequence $\sigma = \sigma_0, \sigma_1, \sigma_2, \dots$, where σ_i is the set of Boolean propositions $\pi \in \Pi$ that are true at position i in the se-

quence. An infinite sequence σ is said to satisfy an LTL formula ϕ (denoted $\sigma \models \phi$) if σ_0 satisfies ϕ as described in (4.2).

$$\begin{aligned}
\sigma_i \models \pi & \quad \text{iff } \pi \in \sigma_i \\
\sigma_i \models \neg\phi & \quad \text{iff } \sigma_i \not\models \phi \\
\sigma_i \models \phi_1 \wedge \phi_2 & \quad \text{iff } \sigma_i \models \phi_1 \text{ and } \sigma_i \models \phi_2 \\
\sigma_i \models \bigcirc\phi & \quad \text{iff } \sigma_{i+1} \models \phi \\
\sigma_i \models \phi_1 \mathcal{U} \phi_2 & \quad \text{iff } \exists k \mid k \geq i \text{ s.t. } \sigma_k \models \phi_2 \\
& \quad \text{and } \forall j \mid i \leq j < k : \sigma_j \models \phi_1
\end{aligned} \tag{4.2}$$

Intuitively, σ_i satisfies the formula $\bigcirc\phi$ if σ_{i+1} satisfies ϕ ; σ_i satisfies the formula $\phi_1 \mathcal{U} \phi_2$ if ϕ_2 becomes true at some future position σ_k , and ϕ_1 is true until then. The operators described in (4.1) and (4.2) can be used to define additional operators. Of note for this work are the disjunction (\vee), implication (\rightarrow), and bi-conditional (\leftrightarrow) operators, which can be derived from the negation (\neg) and conjunction (\wedge) operators, as well as the temporal operators for eventually (\diamond) and always (\square), which can be derived from the until (\mathcal{U}) operator. A more detailed description of the syntax and semantics of LTL formulas can be found in [14].

4.2.2 Controller Synthesis

The work presented here uses a controller synthesis approach that is based on the approach described in [37]. In this approach, the robot's task is defined as a set of LTL formulas over the Boolean propositions $\Pi = \mathcal{X} \cup \mathcal{Y}$, where \mathcal{X} is a set of propositions that represent abstract, time-varying characteristics of the environment and \mathcal{Y} is a set of propositions describing the abstract state of the robot (including its location in the discretized workspace). The set of environment propositions for Example 1 is defined as $\mathcal{X} = \{Target, Closed\}$. The set of robot propositions is defined as

$\mathcal{Y} = \mathcal{R} \cup \{Mark, Stuck\}$, where \mathcal{R} represents the set of regions in the discretized workspace of Figure 4.1.

The desired behavior of the robot is described by a formula ϕ (shown in (4.3)), which is restricted to the General Reactivity (1) class of LTL formulas [51].

$$\phi = (\varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e) \rightarrow (\varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s) \quad (4.3)$$

The formula given in (4.3) defines a desired robot behavior (described by $\varphi_{i,t,g}^s$), in response to the observed behavior of the environment (described by $\varphi_{i,t,g}^e$). The formulas φ_i^e and φ_i^s are Boolean formulas B_i over $\mathcal{X} \cup \mathcal{Y}$, which describe the *initial* states of the environment and robot, respectively. The formulas φ_t^e and φ_t^s , referred to as *safety* specifications, take the form $\Box B_t$, and represent restrictions on the allowable transitions of the environment and robot, which must hold at all times. For φ_t^e , B_t is a Boolean formula over $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc \mathcal{X}$; for φ_t^s , B_t is a Boolean formula over $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc \mathcal{X} \cup \bigcirc \mathcal{Y}$. Finally, the formulas φ_g^e and φ_g^s , referred to as *fairness* and *liveness* conditions, respectively, take the form $\Box \Diamond B_g$, where B_g is a Boolean formula over $\mathcal{X} \cup \mathcal{Y}$, and represent properties of the environment and system which must infinitely often hold true (i.e., goals).

An abridged version of the task specification is shown in (4.4). The environment's initial condition φ_i^e requires that the robot is not sensing a *Target*, and that the door region is not *Closed*. The robot's initial condition φ_i^s requires that it start in the *Base* region, is not *Stuck*, and has not *Marked* a target. The environment safety specification φ_t^e requires that the robot can only find a *Target* in one of the G_i regions. The environment fairness condition φ_g^e specifies that the door region cannot remain *Closed* indefinitely. The robot safety specifications φ_t^s require that the robot avoid getting *Stuck*, that it does not move through the *Door* region when it is *Closed*, and that the robot *Mark* a *Target* (and that it remain *Marked*) after one has been found. Finally, the robot liveness conditions specify that the robot should return to the *Base* after it has

found and *Marked* a *Target*, and should otherwise continuously visit each of the four *Goal* regions.

$$\begin{aligned}
 \varphi_i^e & \left\{ \begin{array}{l} \neg Target \\ \wedge \neg Closed \end{array} \right. \\
 \varphi_t^e & \left\{ \begin{array}{l} \square \left(\neg \bigvee_{i \in \{1,2,3,4\}} G_i \right) \rightarrow \neg \bigcirc Target \end{array} \right. \\
 \varphi_g^e & \left\{ \begin{array}{l} \square \Diamond (\neg Closed) \end{array} \right. \\
 \varphi_i^s & \left\{ \begin{array}{l} \neg Mark \\ \wedge \neg Stuck \\ \wedge Base \end{array} \right. \\
 \varphi_t^s & \left\{ \begin{array}{l} \square (\neg \bigcirc Stuck) \\ \wedge \square (\bigcirc Closed \rightarrow \neg \bigcirc Door) \\ \wedge \square ((\bigcirc Target \vee Mark) \leftrightarrow \bigcirc Mark) \end{array} \right. \\
 \varphi_g^s & \left\{ \begin{array}{l} \square \Diamond (Mark \rightarrow Base) \\ \wedge \bigwedge_{i \in \{1,2,3,4\}} \square \Diamond (\neg Mark \rightarrow G_i) \end{array} \right.
 \end{aligned} \tag{4.4}$$

Note that the second of the robot liveness conditions (φ_g^e) is replaced, during synthesis, with a series of formulas (and corresponding propositions) such that the synthesis process may re-order the goals to a more optimal order; details of this process can be found in [26]. Additionally, the first robot safety condition φ_i^s requires that the robot always avoid being *Stuck*; as such, the proposition *Stuck* is never true in the synthesized controller, and will only become true due to errors in the robot's actuation.

The resulting controller that is synthesized (from the approach described in [51]) takes the form of a finite-state machine (FSM): $\mathcal{A} = \{S, S_0, \Pi, \delta, L\}$. The set of states is given by S , where $S_0 \subseteq S$ is the set of possible initial states, and $\Pi = \mathcal{X} \cup \mathcal{Y}$ is the set

of environment and robot propositions described above. The controller moves between states according to the transition function $\delta : S \times 2^{\mathcal{X}} \rightarrow S$, which maps a state and the set of environment (input) propositions that are true, to a successor state. Finally, $L : S \rightarrow 2^{\mathcal{Y}}$ is a labeling function which maps each state to the subset of the robot propositions \mathcal{Y} that are true in that state.

4.2.3 Optimal Controller Synthesis

This paper utilizes the optimal controller synthesis algorithm introduced in [26], which solves the problem of synthesizing optimal robot controller from high-level task specification for adversarial environment. The controller generated is in the form of finite state machine defined in Section 4.2.2. Each transition in the FSM is assigned with a cost function defined in a tuple (c_d, c_t) , where c_d and c_t can be any value representing the cost of the corresponding transition in the FSM; in [26], c_d and c_t are used to represent the cost of environment-forced delays and the cost of moving between locations in the workspace, respectively.

Given a user-defined cost preference relation, the optimal controller synthesis algorithm can take any prioritization over the possible cost choices and generate the optimal controller to minimize the total accumulated cost over a sequence of states to satisfy the robot goals φ_g^s . The synthesis algorithm uses multiple fixed-point computations to find the set of states that satisfy the given robot tasks. The optimization is achieved in the synthesis step by incorporating the cost metrics in the fixed-point computations. The cost is accumulated for each state as the robot makes progress towards its goals. By choosing the sequence of states with minimal accumulated cost, the synthesized controller is guaranteed to be optimal with respect to the given cost metric.

In this paper, the cost tuple (c_d, c_t) is replaced by a cost c_ψ describing the probabilistic behavior of the robot. The calculation of this cost is detailed in Section 4.4, and

uses a probabilistic model of the system, included the errors in the robot's sensing and actuation.

4.2.4 Probabilistic Error Models

In order to model the effects of sensing and actuation errors on the behavior of the robot, the work in this paper adopts the probabilistic models presented in [28], [29], and [30]. The true changes in the environment configuration are modeled by a set of probabilities $P(X_{t+1}|X_t, Y_t)$, where $X_{t+1} \subseteq \mathcal{X}$ is the set of environment propositions that are true in the next step, $X_t \subseteq \mathcal{X}$ is the set of environment propositions that are true in the current step, and $Y_t \subseteq \mathcal{Y}$ is the set of robot propositions that are true in the current step. Note that, for such a model, the new configuration of the environment is dependent only on the previous configurations of the environment and robot. In the given example, the probability that the robot encounters a target in each of the goal regions is 0.25: $p(Target_{t+1}|G_{i,t}) = 0.25$ for $i \in \{1, 2, 3, 4\}$; the probability that the robot encounters a target elsewhere is 0: $p(Target_{t+1}|Y \in 2^{\mathcal{Y} \setminus \{G_1, G_2, G_3, G_4\}}) = 0$. In any given discrete time step, the probability that the *Door* closes is 0.5: $p(Closed_{t+1}) = 0.5$.

In order to model the robot's sensor error, an additional set of propositions $\overline{\mathcal{X}} ::= \{\overline{x} | x \in \mathcal{X}\}$ is used to represent the robot's observations of the environment propositions, such that each environment proposition $x \in \mathcal{X}$ has a corresponding sensor proposition $\overline{x} \in \overline{\mathcal{X}}$ which, when the sensor is correct, mimics the truth value of the environment proposition. Changes in the sensor configuration, then, are modeled by the set of probabilities $P(\overline{X}_{t+1}|X_{t+1}, \overline{X}_t, Y_t)$. Similarly to the environment model, the new configuration of the sensor $\overline{X}_{t+1} \subseteq \overline{\mathcal{X}}$ is dependent only on the new environment configuration $X_{t+1} \subseteq \mathcal{X}$, and the previous configurations of the sensors $\overline{X}_t \subseteq \overline{\mathcal{X}}$ and robot $Y_t \subseteq \mathcal{Y}$. For the given example, the robot's sensors are described by the set of propositions $\overline{\mathcal{X}} = \{\overline{Target}, \overline{Closed}\}$. The probability that the robot correctly

senses the presence of a *Target* is modeled to be 0.9: $p(\overline{Target}_{t+1}|Target_{t+1}) = p(\overline{-Target}_{t+1}|-Target_{t+1}) = 0.9$. Similarly, the probability that the robot correctly identifies when the *Door* is *Closed* is 0.75: $p(\overline{Closed}_{t+1}|Closed_{t+1}) = p(\overline{-Closed}_{t+1}|-Closed_{t+1}) = 0.75$.

Finally, the changes in the state of the robot are modeled by the set of probabilities $P(Y_{t+1}|\overline{Y}_{t+1}, Y_t)$. This model uses a new set of propositions $\overline{\mathcal{Y}} ::= \{\overline{y}|y \in \mathcal{Y}\}$ to represent the desired state of the robot. That is, the output of the FSM controller is a subset of propositions $\overline{Y}_{t+1} \subseteq \overline{\mathcal{Y}}$ designating the set of robot propositions that *should* become true in the next step. Due to error in the robot's actuation, this set may not exactly match the *actual* configuration of the robot in the next step ($Y_{t+1} \subseteq \mathcal{Y}$). Modeled in this manner, the *actual* new state of the robot $Y_{t+1} \subseteq \mathcal{Y}$ is dependent on the *desired* new state of the robot $\overline{Y}_{t+1} \subseteq \overline{\mathcal{Y}}$, and the current state of the robot $Y_t \subseteq \mathcal{Y}$. In this example, the probability that the robot gets *Stuck* while moving through regions $\{R_4, R_6\}$ (the yellow regions in Figure 4.1) is modeled to be 0.1: $p(Stuck_{t+1}|R_{i,t}) = 0.1$ for $i \in \{4, 6\}$. Similarly, the probability that the robot gets *Stuck* while moving through region R_5 (the red region in Figure 4.1) is 0.35: $p(Stuck_{t+1}|R_{5,t}) = 0.35$. The probability that the robot gets stuck when moving through any other region is modeled to be 0. Note that, in this example, the error in actuation is only depended on the current state of the robot, and is not affected by the desired new state.

It should be noted that this error model does not explicitly account for localization error. Rather, it is assumed that the localization error is small enough that the robot can correctly determine which region (in a discretized workspace) it is in. Any errors in localization, along with errors in the movement of the robot, are captured by the set of probabilities $P(Y_{t+1}|\overline{Y}_{t+1}, Y_t)$ that model the changes in the state of the robot, which includes the robot's location in the discretized workspace.

4.3 Problem Statement

The work presented in this paper describes an approach for synthesizing a robot controller that minimizes the probability that a robot, which operates with errors in its sensing and actuation, will violate a prescribed behavior. Such a process requires that the minimized cost function account for the expected changes in the environment, as well as the expected errors in sensing and actuation. To do this, the changes in the environment, as well as the errors in sensing and actuation, are modeled, as described in Section 4.2.4, with the sets of probabilities $P(X_{t+1}|X_t, Y_t)$, $P(\bar{X}_{t+1}|X_{t+1}, \bar{X}_t, Y_t)$, and $P(Y_{t+1}|\bar{Y}_{t+1}, Y_t)$, respectively.

Additionally, this process requires the specification of a formula which represents the desired behavior of the robot in the probabilistic system. To that end, define the desired behavior of the robot ψ as a Boolean formula over $\mathcal{X} \cup \bar{\mathcal{X}} \cup \mathcal{Y} \cup \bar{\mathcal{Y}}$. This formula is restricted to the form of a Boolean formula over the current and next state of the system (environment, sensors, and robot); in this manner, the synthesis process generates the FSM that minimizes the probability of ever violating the formula ψ , similar to including a safety constraint of the form $\Box\psi$.

The problem addressed in this paper is as follows.

Problem: Given an LTL task specification ϕ , a probabilistic model of the environment $P(X_{t+1}|X_t, Y_t)$, and probabilistic models of the errors in the robot's sensing $P(\bar{X}_{t+1}|X_{t+1}, \bar{X}_t, Y_t)$ and actuation $P(Y_{t+1}|\bar{Y}_{t+1}, Y_t)$, synthesize an FSM \mathcal{A} that satisfies the task specification ϕ and minimizes the probability that a robot (which operates with the modeled probability) will violate the desired behavior ψ . That is, find the FSM \mathcal{A}^* such that $\mathcal{A}^* \models \phi$ and $p(\mathcal{A}^* \| P(\cdot) \not\models \psi) = \min_{\mathcal{A} \models \phi} p(\mathcal{A} \| P(\cdot) \not\models \psi)$, where $P(\cdot)$ denotes the probabilistic models of the environment and sensor/actuator error, and $\mathcal{A} \models \phi$ denotes that the FSM \mathcal{A} satisfies the formula ϕ ($\sigma \models \phi$ for all possible executions σ of \mathcal{A}).

4.4 Cost Function

The work presented in this paper adopts the optimal synthesis approach presented in [26], which is briefly described in Section 4.2.3. This approach synthesizes an FSM that satisfies the robot’s task specification, while minimizing the cost c . For this paper, a different cost function is defined to capture the probability that the robot completes its task while satisfying a given LTL formula. This is represented by the continuous cost c_ψ which captures the probability that the robot violates a given formula ψ when making a particular transition (as prescribed by the controller). Minimization of this cost during the controller synthesis results in a controller that maximizes the probability that the robot satisfies the formula ψ .

For the following, let the current state of the system be represented by $\langle X_t, \bar{X}_t, Y_t \rangle$, which specifies the configuration of the environment $X_t \in \mathcal{X}$, sensors $\bar{X}_t \in \bar{\mathcal{X}}$, and robot $Y_t \in \mathcal{Y}$. The next state of the system, then, is represented by $\langle X_{t+1}, \bar{X}_{t+1}, Y_{t+1} \rangle$, where the actual configuration of the robot is a probabilistic result of the robot attempting to transition to a desired new state $\bar{Y}_{t+1} \in \bar{\mathcal{Y}}$. Following this notation, the cost function evaluates the cost of transitioning from the current sensor and robot configurations $\langle \bar{X}_t, Y_t \rangle$ to a desired robot configuration (as chosen by the controller) for a new sensor configuration $\langle \bar{X}_{t+1}, \bar{Y}_{t+1} \rangle$.

The cost function c_ψ represents the probability that each allowable (with respect to the original specification) transition by the robot results in a new state which violates the formula ψ . A smaller (more negative) value of c_ψ indicates that the chosen transition is less likely to violate the desired behavior ψ , and is therefore a more desirable transition than one with a larger value of c_ψ . Each transition in the controller may result in multiple, different, true states of the system $\langle X_{t+1}, Y_{t+1} \rangle$; it is these true states that determine if the robot’s behavior violates the formula ψ . The transition between two states in the controller is denoted as $\Delta(\langle \cdot \rangle, \langle \cdot \rangle)$, and the cost of that transition is calculated as shown

in (4.5).

$$c_\psi(\Delta(\langle \bar{X}_t, Y_t \rangle, \langle \bar{X}_{t+1}, \bar{Y}_{t+1} \rangle)) = \log \left[\sum_{\langle X_{t+1}, Y_{t+1} \rangle} bel(\langle X_{t+1}, Y_{t+1} \rangle) \cdot (1 - 1_\psi) \right] \quad (4.5)$$

The function 1_ψ is an indicator function, which takes the value 1 when the state-pair $(\langle X_t, Y_t \rangle, \langle X_{t+1}, Y_{t+1} \rangle)$ satisfies the formula ψ , and 0 when the state-pair violates the formula ψ . The summation in (4.5), then, calculates the belief that the specified transition results in a state $\langle X_{t+1}, Y_{t+1} \rangle$ that violates the formula ψ . The synthesis process minimizes the sum of the cost over a sequence of states; by using the log of this belief, this minimization is equivalent to minimizing the product, over the sequence of states, of the belief that the transitions violate ψ .

The component $bel(\langle X_{t+1}, Y_{t+1} \rangle)$, in (4.5), represents the belief that, from a given initial state $\langle X_t, \bar{X}_t, Y_t \rangle$, new sensor configuration \bar{X}_{t+1} , and desired new robot configuration \bar{Y}_{t+1} , the new state of the environment and robot will be $\langle X_{t+1}, Y_{t+1} \rangle$. This belief is found from the model probabilities described in Section 4.3, and given by:

$$bel(\langle X_{t+1}, Y_{t+1} \rangle) = bel(X_{t+1}) \cdot bel(Y_{t+1}) \quad (4.6)$$

where $bel(X_{t+1})$ and $bel(Y_{t+1})$ are given in (4.7) and (4.8), respectively.

$$bel(X_{t+1}) = \eta \cdot p(\bar{X}_{t+1} | X_{t+1}, \bar{X}_t, Y_t) \cdot p(X_{t+1} | X_t, Y_t) \quad (4.7)$$

$$bel(Y_{t+1}) = p(Y_{t+1} | \bar{Y}_{t+1}, Y_t) \quad (4.8)$$

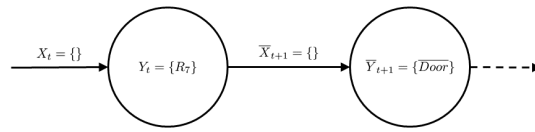
As shown in (4.6), the belief that the new state is $\langle X_{t+1}, Y_{t+1} \rangle$ is broken into the belief that the new environment configuration is X_{t+1} and the new robot configuration is Y_{t+1} ; note that the cost function in (4.5) calculates the summation over the set of possible

states $\langle X_{t+1}, Y_{t+1} \rangle \in 2^{\mathcal{X}} \times 2^{\mathcal{Y}}$. The belief that the new environment configuration is X_{t+1} , shown in (4.7), is calculated from the probabilistic models for the environment and sensor configurations, and is normalized with the normalization constant η . The belief that the new robot configuration is Y_{t+1} , shown in (4.8), is found directly from the probabilistic model for the changes in the configuration of the robot.

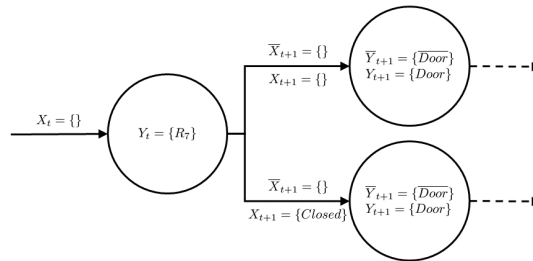
If a particular transition $\Delta(\langle \bar{X}_t, Y_t \rangle, \langle \bar{X}_{t+1}, \bar{Y}_{t+1} \rangle)$ results only in states that satisfy the formula ψ , then the indicator function will be 1 for all of the possible new states, and the cost function (4.5) for that transition will evaluate to $\log(0) = -\infty$. Conversely, if the transition may only result in states that violate ψ , then the indicator function will be 0 for each state and the cost function (4.5) will sum to $\log(1) = 0$. Finally, if a transition results in some states that violate ψ , and some states that satisfy ψ , then the cost function (4.5) will sum to $\log(\beta)$ where β is the total belief that the transition in question results in one of the violating states.

Consider Example 1, where the environment configuration is $X_t = \{\}$ (no *Target* and the door is not *Closed*), and the robot configuration is $Y_t = \{R_7\}$ (the robot is in region R_7 , has not *Marked* a target, and is not *Stuck*). Let the new sensor configuration be $\bar{X}_{t+1} = \{\}$, and let the new desired robot configuration be $\bar{Y}_{t+1} = \{\overline{Door}\}$. Figure 4.2(a) shows a graphical depiction of this transition.

For this configuration, there are two possible new states of the environment $X_{t+1} \in \{\{\}, \{Closed\}\}$ (the door may be either open or *Closed*, and *Target* must be false), and one possible resulting state for the robot $Y_{t+1} = \{Door\}$ (since the robot will not get stuck moving through region R_7). As a result of the possible errors in the sensing and actuation of the robot, the desired transition may result in two possible states of the system: the robot may move into the *Door* region when it correctly senses that it is open, or it may do so when it fails to detect that the door is actually *Closed*. A graphical depiction of the possible transitions is shown in Figure 4.2(b).



(a) Possible desired transition for Example 1, to be evaluated during the controller synthesis.



(b) The possible outcomes of the desired transition.

Figure 4.2: A graphical representation of a single desired transition and the possible outcomes (due to sensing and actuation error) for Example 1.

To find the cost of the intended transition, the belief in each possible outcome is found as described in (4.6)-(4.8). The belief that the transition will result in each of the possible configurations is shown in (4.9). In this equation, η , which normalizes $bel(X_{t+1})$ over the set of possible new environment configurations X_{t+1} , takes a value of 2, such that $(\eta \cdot 0.75 \cdot 0.5) + (\eta \cdot 0.25 \cdot 0.5) = 1$.

$$\begin{aligned}
bel(X_{t+1} = \{\}) &= \eta \cdot p(\neg \overline{Closed}_{t+1} | \neg Closed_{t+1}) \\
&\quad \cdot p(\neg Closed_{t+1}) \\
&= \eta \cdot 0.75 \cdot 0.5 \\
&= 0.75 \\
bel(X_{t+1} = \{Closed\}) &= \eta \cdot p(\overline{Closed}_{t+1} | Closed_{t+1}) \\
&\quad \cdot p(Closed_{t+1}) \\
&= \eta \cdot 0.25 \cdot 0.5 \\
&= 0.25 \\
bel(Y_{t+1} = \{Door\}) &= p(Door_{t+1} | \overline{Door}_{t+1}) \\
&= 1
\end{aligned} \tag{4.9}$$

Given a formula $\psi = \bigcirc Closed \rightarrow \neg \bigcirc Door$ (which requires that the robot avoid entering the *Door* region when it is *Closed*), the indicator function 1_ψ takes a value of 1 when the new state is $\langle X_{t+1}, Y_{t+1} \rangle = \langle \{\}, \{Door\} \rangle$. Conversely, when the new state is $\langle X_{t+1}, Y_{t+1} \rangle = \langle \{Closed\}, \{Door\} \rangle$, the indicator function 1_ψ takes a value of 0 (since the state violates the formula ψ). Accordingly, the evaluation of the cost function for

this possible transition of the FSM is given in (4.10).

$$\begin{aligned}
c_\psi(\Delta(\langle \bar{X}_t, Y_t \rangle, \langle \bar{X}_{t+1}, \bar{Y}_{t+1} \rangle)) &= \\
\log \left[\sum_{\langle X_{t+1}, Y_{t+1} \rangle} \text{bel}(\langle X_{t+1}, Y_{t+1} \rangle) \cdot (1 - 1_\psi) \right] & \quad (4.10) \\
= \log [(0.75 \cdot 1 \cdot (1 - 1)) + (0.25 \cdot 1 \cdot (1 - 0))] & \\
= \log(0.25) &
\end{aligned}$$

Formulated in this manner, the cost function described in Eqs. 4.5-4.8 is reminiscent of Bayesian estimation and filtering techniques. The likelihood of each new *true* state $\langle X_{t+1}, Y_{t+1} \rangle$ is determined from the previous state $\langle X_t, Y_t \rangle$, the control inputs \bar{Y}_{t+1} , and the new sensor information \bar{X}_{t+1} . The new state is determined using the prior probabilities that model the changes in the environment $p(X_{t+1}|X_t, Y_t)$ and robot $p(Y_{t+1}|\bar{Y}_{t+1}, Y_t)$ configurations, as well as the posterior probability that models the changes in the robot's sensors $p(\bar{X}_{t+1}|X_{t+1}, \bar{X}_t, Y_t)$. Note that the proposed cost function also assumes the Markov property: the probability distribution of the new state is independent of the state history, given the current state.

It is important to note that the current environment configuration X_t in this cost function is assumed to be equivalent to the current sensor configuration \bar{X}_t . That is, the cost function assumes that the current state is a “correct” one: the state does not violate any specifications, and the sensor configuration is an accurate representation of the environment. As a whole, this means that synthesis algorithm optimizes the probability that a robot following the prescribed behavior will satisfy the formula ψ . In other words, this method does not optimize the controller's chance of recovering from an incorrect (but safe) transition while satisfying ψ .

4.5 Results

4.5.1 Changing ψ

Several controllers were synthesized, using the approach discussed in Section 4.2.3 and minimizing the cost c_ψ , for Example 1. The behavior of the synthesized controller can be changed by choosing a different formula ψ to use for the cost function c_ψ . The result is a controller which minimizes the probability that the robot violates the prescribed behavior; this allows one to easily change the emphasized probabilistic behavior of the robot to match a desired task. Figure 4.3 shows the resulting paths, which are truncated, for the sake of clarity, to the portion of the behavior from the robot's initial location (shown with a solid black circle) to the robot's position after visiting each of the four goal regions once (hollow circle), for three different controllers, synthesized with different desired behavior formulas ψ .

Additionally, the probabilistic behavior of each of the controllers was analyzed to find the probability that the robot, when operating with the model probabilities described in Section 4.2.4, will find the target and return to the *Base* without getting stuck or entering a closed *Door* region. These probabilities are found using the approach presented in [29] and [30] to compose a discrete, probabilistic model of the system that includes sensor and actuation error. The PRISM probabilistic model checking software [38] is then used to find the probability that the composed system satisfies the following LTL formula, which requires that the system maintains the property $\Box(\neg Stuck \wedge (Closed \rightarrow \neg Door))$.

Figure 4.3(a) shows the truncated path of a robot operating with a controller found by minimizing the formula $\psi = \bigcirc Closed \rightarrow \neg \bigcirc Door$. In this case, the controller is synthesized to minimize only the probability that the robot erroneously enters the *Door* region when it is *Closed*, and is not required to avoid the regions where it may

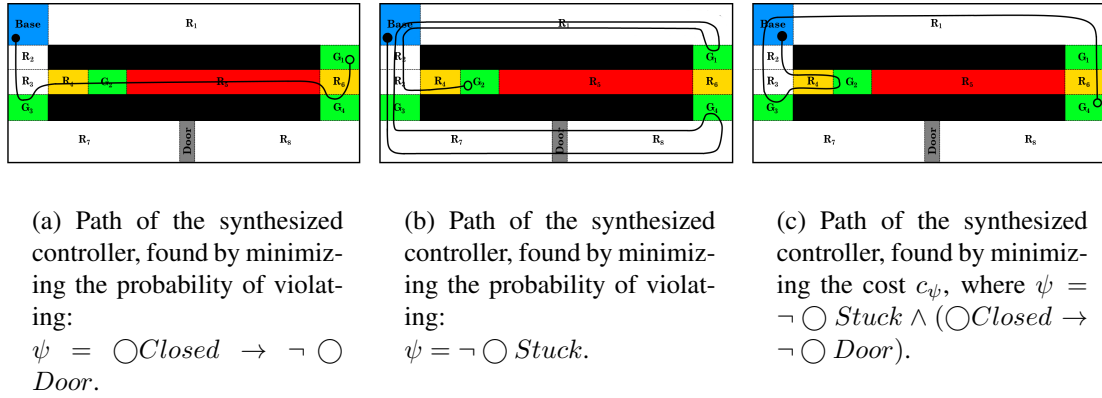


Figure 4.3: Resulting paths for three controllers that were synthesized with different ψ formulas. While the synthesized controllers exhibit infinite behaviors, the depicted paths are truncated to the initial visit to each goal region $G_1 - G_4$.

get *Stuck*. The resulting controller avoids the *Door* region, and takes the shortest path to exploring each of the four goal regions. This controller results in a probability of 0.4558 that the robot satisfies $\Box(\neg Stuck \wedge (Closed \rightarrow \neg Door))$. In contrast, because this controller completely avoids the *Door* region, the robot has a probability of 1.0 that it will maintain the behavior described by ψ and satisfy $\Box(Closed \rightarrow \neg Door)$.

Figure 4.3(b) shows the truncated path of a robot operating with a controller found by minimizing the formula $\psi = \neg \bigcirc Stuck$. This controller proceeds through the *Door* region, in order to avoid the regions R_5 and R_6 where it may get *Stuck*. It does so because the new formula ψ only penalizes the controller for getting *Stuck*, but no longer does so for the robot entering the *Door* region when it is *Closed*. Furthermore, note that the synthesized controller approaches G_2 by traversing through R_4 , which has a lower probability of resulting in the robot getting stuck than traversing through region R_5 . The controller synthesized for this desired behavior ψ improves upon the probabilistic behavior of the first controller, and has a probability of 0.5298 of completing its task without getting *Stuck* or entering the *Door* region when it is closed. In contrast, the robot has a probability of 0.9325 that it will maintain the behavior described by ψ and satisfy $\Box(\neg Stuck)$.

Finally, a third controller was synthesized for a formula ψ that requires the robot to avoid getting stuck and to only proceed through the *Door* region when it is not closed: $\psi = \neg \bigcirc Stuck \wedge (\bigcirc Closed \rightarrow \neg \bigcirc Door)$. Figure 4.3(c) shows the truncated path for the robot. The behavior exhibited by this controller takes a more complicated path, in order to avoid the region R_5 , as well as the *Door* region. This is due to the high probability of violating ψ that is introduced when traversing each region. Region R_5 exhibits a relatively high probability (0.35) that the robot gets stuck, while there is a reasonably large probability (0.25) that the robot fails to correctly sense that the *Door* is *Closed* and enters the region; in either case, the robot violates the prescribed formula ψ . In contrast, the regions R_4 and R_6 , which must be traversed in order to reach regions G_2 and G_4 (if the robot is avoiding *Door* and R_5) have a considerably lower probability (0.1) of causing the robot to violate ψ . This controller improves upon the probabilistic behavior of both of the prior controllers (with respect to the analyzed property), and has a probability of 0.7125 that it satisfies the property $\Box(\neg Stuck \wedge (Closed \rightarrow \neg Door))$.

4.5.2 Changing the Model Probabilities

In addition to modifying the desired behavior ψ , new behaviors can be found when controllers are synthesized for systems with different model probabilities $P(X_{t+1}|X_t, Y_t)$, $P(\bar{X}_{t+1}|X_{t+1}, \bar{X}_t, Y_t)$, and $P(Y_{t+1}|\bar{Y}_{t+1}, Y_t)$. Consider the controller that was synthesized for $\psi = \neg \bigcirc Stuck \wedge (\bigcirc Closed \rightarrow \neg \bigcirc Door)$, that resulted in the path shown in Figure 4.3(c). If a different set of model probabilities are used for the controller synthesis, the resulting controller will minimize the probability of violating ψ for the new probabilities.

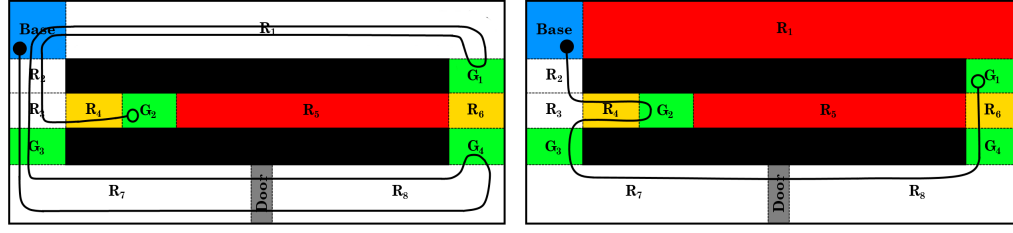
If the probabilistic model of the system is adjusted such that the *Door* is unlikely to become *Closed* by adjusting the probability $p(Closed_{t+1}) = 0.1$, then the synthesized controller yields a different behavior (despite being synthesized for the same ψ). Fig-

ure 4.4(a) shows the resulting (truncated) path for a controller synthesized with this new probabilistic model and a desired behavior $\psi = \neg \bigcirc Stuck \wedge (\bigcirc Closed \rightarrow \neg \bigcirc Door)$. Because the *Door* is now unlikely to actually *Close*, the robot can more safely move through that region; as such, this controller exhibits the same behavior as the controller shown in Figure 4.3(b), where the desired behavior ψ did not account for a *Closed Door*. For a system exhibiting the modified probabilistic behavior, the new controller has a probability of 0.8595 of satisfying $\Box(\neg Stuck \wedge (Closed \rightarrow \neg Door))$; in contrast, the original controller is less likely to satisfy this property (probability of 0.7125) in the modified system.

Similarly, if the probabilistic model is instead adjusted such that the robot is highly likely to get *Stuck* when moving through region R_1 (let $p(Stuck_{t+1}|R_{1,t}) = 0.35$ and $p(Closed_{t+1}) = 0.5$), a different behavior is exhibited for $\psi = \neg \bigcirc Stuck \wedge (\bigcirc Closed \rightarrow \neg \bigcirc Door)$. Figure 4.4(b) shows the resulting (truncated) path for this controller. Note that the controller now avoids region R_1 , even at the expense of moving through the *Door* region and region R_6 . The new controller has a probability of 0.5855 that it satisfies $\Box(\neg Stuck \wedge (Closed \rightarrow \neg Door))$ for the modified model probabilities, while the original controller, which was synthesized without accounting for the increased probability of getting *Stuck* while moving through R_1 , performs worse, with a probability of 0.5668.

4.6 Conclusion

This paper presents a method for automatically synthesizing a robot controller from a set of high-level task specifications, which maximizes the probability that a robot, operating under the modeled conditions, will satisfy a given temporal logic formula. The presented work uses the synthesis approach described in [26], and gives a cost function that captures the probability that a robot will violate a formula ψ . By minimizing this



(a) Path of the synthesized controller, when the probabilistic model is adjusted such that $p(Closed_{t+1}) = 0.1$.

(b) Path of the synthesized controller, when the probabilistic model is adjusted such that $p(Stuck_{t+1}|R_{1,t}) = 0.35$.

Figure 4.4: Resulting paths for two controllers that were synthesized for different probabilistic models, with $\psi = \neg \bigcirc Stuck \wedge (\bigcirc Closed \rightarrow \neg \bigcirc Door)$. While the synthesized controllers exhibit infinite behaviors, the depicted paths are truncated to the initial visit to each goal region $G_1 - G_4$.

cost during the controller synthesis, the resulting controller maximizes the probability that the robot satisfies the formula throughout execution.

This work complements the authors' previous work [28–30], and relaxes the assumption that a robot's sensors and actuators operate without error. By relaxing this assumption and probabilistically modeling the behavior of the environment and errors in the sensors and actuators, the proposed method can be used to automatically create the deterministic controller that is most likely to behave in the desired manner. This allows the user to quickly and easily create robot controllers for complex tasks, which are guaranteed to behave in a manner that minimizes the probability that the robot will violate a prescribed behavior, under the modeled probabilities. Coupled with the authors' previous work, this provides a method for synthesizing, analyzing, and revising correct-by-construction robot controllers in situations where the robot operates with errors in its sensing and actuation.

Future extensions of this work may investigate extending the cost-function to allow for the placement of constraints on the total cost, allowing for the synthesis of a controller which minimizes a given cost function (such as the distance traveled) while con-

strained by another cost function (such as the probability that the robot satisfies a given formula). Another area for future investigation is a modification to the cost function or synthesis procedure that explicitly plans for the robot to take unintended transitions; such a modification would allow the controller to robustly recover from errors during execution.

CHAPTER 5

**EXPERIMENTAL EVALUATION AND FORMAL ANALYSIS OF
HIGH-LEVEL TASKS WITH DYNAMIC OBSTACLE ANTICIPATION ON A
FULL-SIZED AUTONOMOUS VEHICLE**

5.1 Introduction

Complex robotic tasks require solutions that integrate a large number of different components to perceive and interpret the robot’s environment, and to decide and act intelligently based on those perceptions. An autonomous vehicle, for example, must not only localize itself within its environment, but must also locate other vehicles and obstacles, and act to avoid them. In most cases, a robot’s perception of its environment is represented probabilistically, while the robot’s decision making must necessarily be deterministic.

Autonomous vehicles typically use planners that rely on rapid re-planning to react to dynamic objects in the environment [3, 47, 58]. In order to drive safely around other vehicles, these planners use ad-hoc rules to decide whether or not an action is safe [5]. The brittleness of this approach was exemplified by the collision between the Cornell and MIT 2007 DARPA Urban Challenge entries [19]. A natural, formal way to improve autonomous vehicle safety when operating in dynamic environments is to include ‘anticipation’, or explicitly reasoning about the future behavior of dynamic objects, in planning algorithms. Anticipation of dynamic objects has received interest in the field in recent years, and fully probabilistic, fast, and accurate algorithms are available [6, 13, 21, 23, 25, 48, 63]. Integration of anticipation and planning can be done at a low-level, by including anticipation as a term in a cost-function while computing vehicle actuation commands, as in [23], or at a high-level, using anticipation to make route and behavior decisions, as in previous work by the authors [27].

One method for creating controllers for complex tasks is to formally synthesize a discrete controller from a set of high-level task specifications [4, 8, 17, 32, 34, 36, 44, 56, 61]. Such methods typically use temporal logic formulas to specify the desired behavior of the robot, and allow for the creation of controllers that are provably correct; that is, the synthesized controllers are guaranteed to satisfy the task specifications from which they were synthesized.

The work presented in this paper adopts the approach described in [37], in which a temporal logic task specification is used to describe the desired behavior of a robot, in reaction to perceived changes in the environment. This approach can then be used to automatically generate a correct-by-construction controller that is guaranteed to satisfy the underlying task specification. This guarantee, however, is predicated on the assumption that the robot operates with perfect sensing and actuation.

This paper continues previous work [27], and investigates the fusion of probabilistic perception (through the anticipation of dynamic obstacles) with deterministic decision making (by a correct-by-construction hybrid controller). The anticipation algorithms provide a probabilistic belief about the future state of the environment, allowing the robot to plan ahead rather than relying entirely on reactionary planning. The formal controller synthesis allows for the creation of controllers that behave in a verifiable manner; furthermore, given a probabilistic model of the environment and the system, the probabilistic behavior of the robot can also be formally verified [28].

The work discussed in this paper applies this approach to the problem of an autonomous vehicle operating in environment with multiple intersections and other moving vehicles. Unlike the authors' previous work [27], in this paper the anticipation algorithm and synthesized controller are implemented on a full-scale autonomous car, pictured in Figure 5.1, which operated in a controlled environment with multiple simulated vehicles, which were controlled by human operators (also shown in Figure 5.1).



Figure 5.1: The autonomous vehicle, operating with a high-level, synthesized controller, and anticipating the behavior of multiple, simulated obstacle-vehicles, which were controlled by human drivers (also pictured).

Furthermore, experimental data was collected using the autonomous vehicle and human-controlled obstacles (rather than via simulation), and the results are compared to a formal analysis of the synthesized controller. The portions of this chapter that are original work by the author are the

It should be noted that this chapter presents work that was conducted in collaboration with Frank Havlak, another Ph.D. Candidate in Mechanical Engineering at Cornell University. The controller synthesis, the formal model of the system, and the formal analysis and comparison with the experimental data were all completed by the author of this thesis; the obstacle anticipation, the abstraction of that anticipation to a Boolean sensor value, and much of the implementation to run the code on the vehicle itself was completed by Frank Havlak. Both authors collaborated to conduct the experiment, with the assistance of several colleagues who helped to operate the obstacle vehicles.

5.1.1 Related Work

Some recent research has investigated the use of synthesized controllers under conditions where the sensors or actuators cannot be assumed to be perfect. In [42] and [43], the authors present a method for incorporating a stochastic measure of the error in the robot's motion into the synthesis process, in order to generate the motion plan that is most likely to satisfy a temporal logic specification. In [45], the authors use random variables to represent uncertainty in the characterization of a robot's environment, and use Process Algebra to probabilistically validate behavior-based controllers and provide guarantees on the performance of the robot. In [57], the authors present an approach for synthesizing a controller for a robot which operates in the presence of other, probabilistic agents; the synthesized controller maximizes the probability that the robot will satisfy a given temporal logic specification.

In [61] the authors apply a receding-horizon approach to the problem of controller synthesis, and present the approach in the context of an autonomous vehicle. The resulting method allows the vehicle to quickly re-plan to account for unexpected obstacles and events during operation; by applying a receding-horizon approach, the scope of the synthesis is restricted enough to allow for online re-planning. In [2], the authors present an approach for the online verification of trajectories for autonomous vehicles, using reachability analysis to guarantee the safety of the maneuvers, in the presence of other, moving vehicles.

Incorporating anticipation of dynamic objects in robot planning has seen increased interest in recent years. The problem is complex; it is inherently probabilistic because perfect knowledge of the future is impossible, and it requires not only a model of the dynamics of the object being anticipated, but also of the controller of that object. Some works simplify the problem by making deterministic predictions of object behavior [6, 48, 50], and these approaches work well when the behavior of the dynamics objects is

known or communicated to the robot, as in a cooperative scenario.

Anticipation approaches that account for uncertainty in the future actions of dynamic objects tend to resemble recursive estimation filters (like the Kalman filter and its variants), as in [12, 21]. These approaches can be extended by using Gaussian Mixture Models (GMMs) to relax Gaussian assumptions and reduce linearization errors, and to include discrete states to capture high-level decisions of dynamic objects, as in [25]. The assumed controller models can be improved using data-driven machine learning tools, as in [24].

The work presented in this paper directly addresses the problem of the incorporation of probabilistic environmental data into a deterministic, synthesized controller, and provides probabilistic guarantees on the performance of the system. Furthermore, these guarantees are compared to experimental data obtained from the operation of a full-scale autonomous vehicle, operating in the presence of human-controlled obstacle vehicles.

This paper is structured as follows. Section 5.2 presents several background definitions. Section 5.3 describes the technical approach and the problem formulation. Sections 5.4 and 5.5 describe the methodology used for the formal analysis and experimental platform, respectively. Section 5.6 presents and compares the results of the analysis and experiment. Concluding remarks are given in Section 5.7.

5.2 Preliminaries

5.2.1 Linear Temporal Logic

Linear Temporal Logic (LTL) is a logical formalism that allows for the expression of temporal properties, over a set of Boolean propositions Π . The syntax for an LTL formula ϕ is given in Equation 5.1. Such a formula consists of propositions ($\pi \in \Pi$),

logical operators (\neg , \wedge), and temporal operators (\bigcirc , \mathcal{U}).

$$\phi ::= \text{true} \mid \pi \in \Pi \mid \neg\phi \mid \phi \wedge \phi \mid \bigcirc \phi \mid \phi \mathcal{U} \phi \quad (5.1)$$

The semantics of an LTL formula are defined over an infinite sequence $\sigma = \sigma_0, \sigma_1, \sigma_2 \dots$, where σ_i consists of a set of truth assignments to the Boolean propositions $\pi \in \Pi$. The semantics of an LTL formula are defined in Equation 5.2, where $\sigma_i \models \phi$ denotes that σ_i satisfies the formula ϕ (conversely, $\sigma_i \not\models \phi$ denotes that σ_i does *not* satisfy the formula ϕ). A sequence σ , then, satisfies a formula ϕ if and only if $\sigma_0 \models \phi$.

$$\begin{aligned} \sigma_i \models \pi & \quad \text{iff } \pi \in \sigma_i \\ \sigma_i \models \neg\phi & \quad \text{iff } \sigma_i \not\models \phi \\ \sigma_i \models \phi_1 \wedge \phi_2 & \quad \text{iff } \sigma_i \models \phi_1 \text{ and } \sigma_i \models \phi_2 \\ \sigma_i \models \bigcirc\phi & \quad \text{iff } \sigma_{i+1} \models \phi \\ \sigma_i \models \phi_1 \mathcal{U} \phi_2 & \quad \text{iff } \exists k \mid k \geq i \text{ s.t. } \sigma_k \models \phi_2 \\ & \quad \text{and } \forall j \mid i \leq j < k : \sigma_j \models \phi_1 \end{aligned} \quad (5.2)$$

For the sake of brevity, only the portions of the LTL semantics relevant to the topic of this paper are described here; for a more complete definition of the syntax and semantics of LTL, the reader is referred to [14]. Intuitively, the operator \neg indicates negation, and the operator \wedge indicates a conjunction. The additional operators \vee (disjunction), \rightarrow (implication), and \leftrightarrow (bi-conditional) can be defined from these two operators, and are used to allow for formulas to be written in a more concise manner. The temporal operator \bigcirc denotes that a formula must be true in the *next* position in the sequence, while the operator \mathcal{U} indicates that some formula ϕ_1 must be true for all positions in a sequence up *until* a position where another formula ϕ_2 is true. These temporal operators can similarly be used to define additional operators; of note for this paper are the operators \Box (which requires that a formula is true at all positions in a sequence) and \Diamond (which denotes that a formula must be true for some future position in the sequence).

5.2.2 Controller Synthesis

The work presented in this paper adopts the approach described in [37] for automatically synthesizing a correct-by-construction, discrete controller from a set of temporal logic specifications. This approach uses a discrete abstraction of the system, where the state of the environment is represented by a set of Boolean propositions $\mathcal{X} = \{x_1, x_2, \dots, x_n\}$ that model abstract events, which the robot observes. In the context of the autonomous vehicle problem used in this paper, the environment propositions are used to represent whether or not each of the vehicles route choices (at an intersection) are safe; this is done with a set of Boolean propositions $Unsafe_i$, which are true when a particular route is *unsafe* and false when that route is *safe*.

Similarly, the state of the robot is represented by the set of abstract propositions $\mathcal{Y} = \{a_1, \dots, a_l, r_1, \dots, r_m\}$, which models the abstract actions that the robot can perform (a_i propositions) as well as the location of the robot within a discretized workspace (r_i propositions). For the autonomous vehicle example problem, the set of location propositions r_i represent which road segment the vehicle is on, in a discretized map. The a_i action propositions are used to track the decisions that the vehicle makes at intersections: this includes a proposition that is true when the vehicle *Stops*, as well as a set of propositions $Drive_i$ that indicate which route the vehicle takes when traversing the intersection, where those routes correspond to the set of environment propositions $Unsafe_i$.

From these abstractions and a task specification, a discrete controller (in the form of a finite state automaton) can be synthesized, which is guaranteed to satisfy the given task specification. The task specification is given in a fragment of LTL known as General Reactivity (1), which is described, in detail, in [51]. LTL formulas in this fragment take the form given in Equation 5.3.

$$\phi = (\varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e) \rightarrow (\varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s) \quad (5.3)$$

The above formula specifies a desired behavior for the robot $\varphi^s = \varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s$, in response to changes in the state of the environment $\varphi^e = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e$. In each case (environment and robot) the behavior is broken down further, into three parts: an *initial* condition $\varphi_i^{e,s}$, *safety* specifications $\varphi_t^{e,s}$, and *fairness/liveness* conditions $\varphi_g^{e,s}$. The initial condition is given as a set of Boolean formulas B_i over the propositions $\mathcal{X} \cup \mathcal{Y}$, and restricts the set of allowed initial configurations for the environment and the robot. The safety specifications are given in the form $\Box B_t$, where B_t is a Boolean formula over $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc \mathcal{X} \cup \bigcirc \mathcal{Y}$; these formulas restrict the allowable transitions of the environment and the robot. The fairness conditions φ_g^e represent conditions that the environment must satisfy infinitely often (e.g., specifying that a door in the environment must become open at some point in the future), while the liveness conditions φ_g^s represent goals that the robot is required to satisfy; both conditions take the form $\Box \Diamond B_g$, where B_g is a Boolean formula over $\mathcal{X} \cup \mathcal{Y}$.

From this specification, and the abstractions of the environment and robot, a controller is synthesized using the approach described in [51]. The synthesized controller takes the form of a finite state automaton $\mathcal{A} = \{S, S_0, \mathcal{X}, \mathcal{Y}, \delta, L\}$, with a set of states S and a set of initial states $S_0 \subseteq S$. The sets of propositions \mathcal{X} and \mathcal{Y} represent the abstracted state of the environment and robot, as described previously. The transition function $\delta : S \times 2^{\mathcal{X}} \rightarrow S$ maps each state and set of environment (input) propositions to a successor state. Finally, the labeling function $L : S \rightarrow 2^{\mathcal{Y}}$ maps each state to the set of robot propositions that are true at that state.

5.2.3 Gaussian Mixture Models

The anticipation algorithm used in this work represents the probability density function (pdf) over the state of a tracked dynamic object using Gaussian Mixture Models (GMMs) [1]. GMMs are frequently used to represent non-Gaussian pdfs because they

retain the convenient computational properties of Gaussian distributions, but can be used to represent arbitrarily shaped pdfs. The pdf over a tracked dynamic object's state at time k (\mathbf{x}_k) is given by:

$$p(\mathbf{x}_k) = \sum_{i=1}^{N_k} w_k^i \cdot \mathcal{N}(\mathbf{x}_k | \mu_k^i, \Sigma_k^i) \quad (5.4)$$

where w_k^i , μ_k^i , and Σ_k^i are the weight, mean, and covariance of the i^{th} mixand in the GMM, and N_k is the number of mixands in the GMM at time k . When propagating the pdf forward in time through a dynamics model, each mixand can be treated independently, so any of the many tools for propagating Gaussian uncertainties (e.g., Extended Kalman transform, Sigma Point transform, etc.) can be used.

5.3 Technical Approach and Problem Formulation

In this work, the goal is to examine the fusion of probabilistic perception with deterministic decision making. The behavior of a synthesized controller is formally verified with respect to the probabilistic anticipation of dynamic obstacles in the robot's environment. This verification is used to examine the tradeoffs of the abstraction of the probabilistic information, and the results are compared to experimental data.

An overview of the approach presented in this paper is shown in Figure 5.2. First, a high-level controller is synthesized from a set of task specifications. This controller is then used to control the robot during execution, by making decisions based on the anticipated behavior of dynamic obstacles in the robot's environment. Before the results of the anticipation are passed to the high-level controller, they are first abstracted from a set of probability distributions to a set of Boolean values characterizing the robot's environment. In addition to the execution of the controller on the robot, a formal analysis is performed on a composed model of the controller and the system probabilities, to

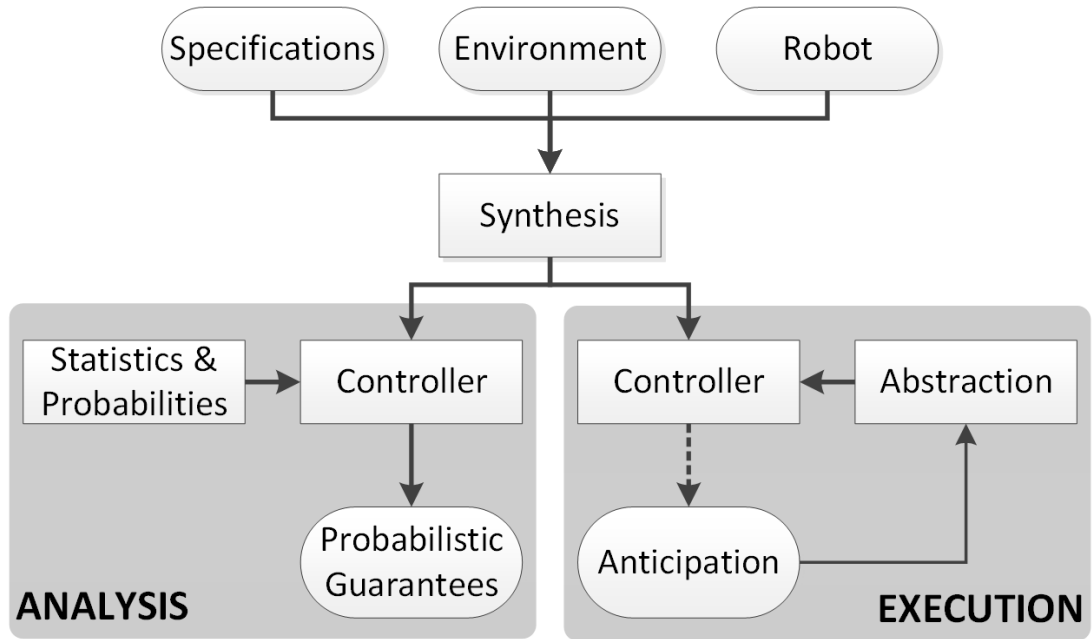


Figure 5.2: Overview of the approach presented in this paper.

provide probabilistic guarantees on how the robot will behave during execution. The approach is applied, in this paper, to the problem of an autonomous vehicle navigating a road system while avoiding other vehicles.

5.3.1 Scenario

For the scenario used as an example in this paper, an autonomous vehicle is tasked with navigating through the environment shown in Figure 5.3. The vehicle must repeatedly visit each of the four corners on the map (marked in blue), in numerical order, and it must avoid colliding with three dynamic obstacles (other vehicles), which are traveling around the same workspace. When the vehicle arrives at an intersection (marked in red), it determines which of the available choices are *safe* and which are *unsafe*. The map used for the scenario examined in this paper is shown in Figure 5.3, though the approach can be more generally applied to a wide variety of robotics scenarios. The approach presented here could also be directly applied to a different road system, where

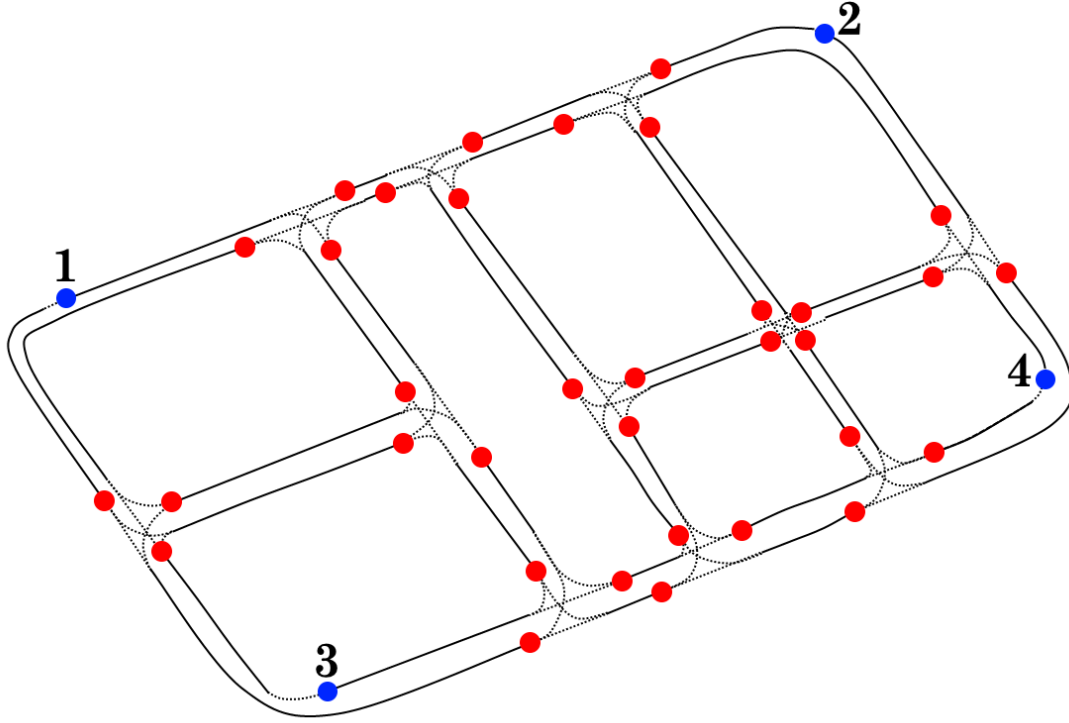


Figure 5.3: Map of the workspace, divided into road segments. Intersections are indicated by circles, with the possible vehicle choices shown as dashed lines. The vehicle is required to repeatedly visit the four corners of the map, which are numbered and marked with the blue circles.

the roads can be discretized into separate segments with known interconnections.

To accomplish this, the environment is abstracted to a set of Boolean propositions $\mathcal{X} = \{Unsafe_1, Unsafe_2, Unsafe_3\}$. Each proposition is used to indicate that a particular choice at the vehicle's current intersection is likely to result in a collision with one of the dynamic obstacles (and is therefore *unsafe*). This allows the vehicle to use the probabilistic anticipation of the dynamic obstacles to determine which roads it may safely traverse (this process is discussed in Sections 5.3.3 and 5.3.4).

Furthermore, these propositions are restricted by layout of the workspace, such that an unavailable choice at an intersection is automatically deemed *unsafe*. That is, if the vehicle arrives at an intersection where there are only two outgoing roads, $Unsafe_3$ is automatically set to *true*; likewise, if there is only one outgoing road at an intersection,

both $Unsafe_2$ and $Unsafe_3$ are automatically set to *true*.

5.3.2 Synthesized Controller

To synthesize a controller for this scenario (using the approach described in Section 5.2.2), a set of propositions $\mathcal{Y} = \{Stop, Drive_1, Drive_2, Drive_3, r_1, \dots, r_m\}$ is defined to model the abstract state of the vehicle. The set of propositions $\{r_1, \dots, r_m\}$ are mutually exclusive propositions which represent the location of the vehicle in the discretized workspace (i.e., which road segment the vehicle is on). The propositions $\{Stop, Drive_1, Drive_2, Drive_3\}$, while not necessary for the synthesis and operation of the controller, are used to indicate which decision is made by the controller at an intersection; the inclusion of these propositions allows the human operator to more easily observe the robot's decisions, and simplifies the specification of the robot's task. The proposition $Drive_1$ indicates that the vehicle chooses to drive on the road segment corresponding to the $Unsafe_1$ environment proposition; likewise, $Drive_2$ and $Drive_3$ correspond to $Unsafe_2$ and $Unsafe_3$ respectively. The proposition $Stop$ indicates, when true, that the vehicle chooses to stop at the intersection and wait for its desired path to become safe.

The vehicle's controller is synthesized from an LTL task specification (as described in Section 5.2.2) over the sets of propositions representing the abstracted state of the environment and the robot (\mathcal{X} and \mathcal{Y} , respectively). An abridged specification is shown below, where $out(r_j)$ is used to represent the set of outgoing roads from road segment r_j .

1. $\neg Unsafe_1 \wedge \neg Unsafe_2 \wedge \neg Unsafe_3$
2. $\neg Stop \wedge \neg Drive_1 \wedge \neg Drive_2 \wedge \neg Drive_3$
3. $\Box(r_j \rightarrow (\bigcirc r_j \vee \bigvee_{r_k \in out(r_j)} \bigcirc r_k))$

4. $\Box \bigwedge_{i \in \{1,2,3\}} (Unsafe_i \rightarrow \neg Drive_i)$
5. $\Box \bigwedge_{r_k \in out(r_j)} ((r_j \wedge \bigcirc r_k) \leftrightarrow \bigcirc Drive_i)$
 where r_k is the i^{th} road segment in $out(r_j)$
6. $\bigwedge_g \Box \Diamond r_g$

The first and second formulas restrict the allowable initial conditions for the environment and robot, respectively. The third formula restricts the motion of the vehicle such that it may only transition to a new road segment r_k if that segment is an outgoing road from the vehicle's current location r_j (the vehicle may also remain in its current location r_j). The fourth formula requires that, if a particular road choice is deemed unsafe, the vehicle is not allowed to drive on it. The fifth formula describes a set of formulas which map the road segment transitions (i.e. moving from segment r_j to segment r_k) to their corresponding $Drive_i$ propositions; when taken in conjunction with the fourth formula, this restricts the vehicles movement to only those roads that are deemed *safe*. Finally, the sixth formula requires that the vehicle repeatedly visit all of the goal locations, providing the liveness conditions for the synthesized controller.

5.3.3 Dynamic Obstacle Anticipation

This work anticipates the future states of tracked vehicles in the environment to evaluate the safety of candidate robot actions, and arrive at the Boolean safety propositions \mathcal{X} . In order to anticipate the states ($\mathbf{x}_{k,obs}$) of tracked vehicles, a simple dynamics model is assumed:

$$\mathbf{x}_{k+1,obs} = f(\mathbf{x}_{k,obs}). \quad (5.5)$$

In this case, tracked vehicles are modeled as four-state bicycle robots, and the dynamics model f consists of a bicycle dynamics model composed with simple lane-following

and speed-keeping controllers. The high-level decision making of tracked objects is modeled by assuming that the tracked object makes decisions randomly with uniform probability (e.g., an object approaching a four-way intersection is assumed to decide to turn left, go straight, and turn right with equal probability).

Both the bicycle-dynamics vehicle model and simple controller model are nonlinear, so the true pdf over the tracked vehicle's state will become progressively more non-Gaussian as it is propagated forward in time. The anticipation algorithm used in this paper uses GMMs to model the non Gaussian pdfs over the tracked vehicle's state, automatically detects and adapts the GMM to nonlinearities in the vehicle and controller models, and captures the high-level decisions of the tracked vehicle by including discrete states (for example, which lane or road the tracked vehicle is following) [25]. Given an estimate of the state of a tracked object at time k ($\mathbf{x}_{k,\text{obs}}$), from tracking algorithms, the anticipation algorithm calculates the pdf over the tracked vehicle's state at future time steps $k + 1$ through $k + H$, where H is the anticipation horizon:

$$\mathcal{N}(\mathbf{x}_{k,\text{obs}}|\mu_{k,\text{obs}}, \Sigma_{k,\text{obs}}) \xrightarrow{f} \{p(\mathbf{x}_{k+1,\text{obs}}), \dots, p(\mathbf{x}_{k+H,\text{obs}})\} \quad (5.6)$$

where $p(\mathbf{x}_{k+j,\text{obs}})$ is equal to:

$$p(\mathbf{x}_{k+j,\text{obs}}) = \sum_{i=1}^{N_{k+j}} w_{k+j}^i \cdot \mathcal{N}(\mathbf{x}_{k+j,\text{obs}}|\mu_{k+j,\text{obs}}^i, \Sigma_{k+j,\text{obs}}^i). \quad (5.7)$$

This probability distribution is computed by propagating the initial distribution through the assumed dynamics model many times using the sigma-point transform [31]. The anticipation algorithm used automatically detects accuracy errors due to nonlinearities in the assumed model, and adapts the mixture accordingly by splitting individual mixands. A detailed description is available in [25]. Section 5.3.4 describes how the anticipation result is used to compute the safety of a candidate action.

5.3.4 Sensor Abstraction

The anticipated tracked vehicle trajectories must be further abstracted to arrive at the Boolean safe/unsafe environment propositions \mathcal{X} .

To evaluate the safety of a candidate action, the robot projects its own behavior forward in time for that candidate action, and computes the probability of collision between itself executing that action and the anticipated behavior of tracked obstacles in the environment, up to an anticipation horizon. To arrive at a Boolean valued result, a safety threshold on the probability of collision is defined, above which the candidate action is *unsafe* and below which the candidate action is *safe*.

For the j^{th} candidate action k timesteps in the future, the probability of collision is defined:

$$p_{\text{coll}}(\mathbf{x}_{k,\text{robot}}^j, \mathbf{x}_{k,\text{obs}}) \quad (5.8)$$

where

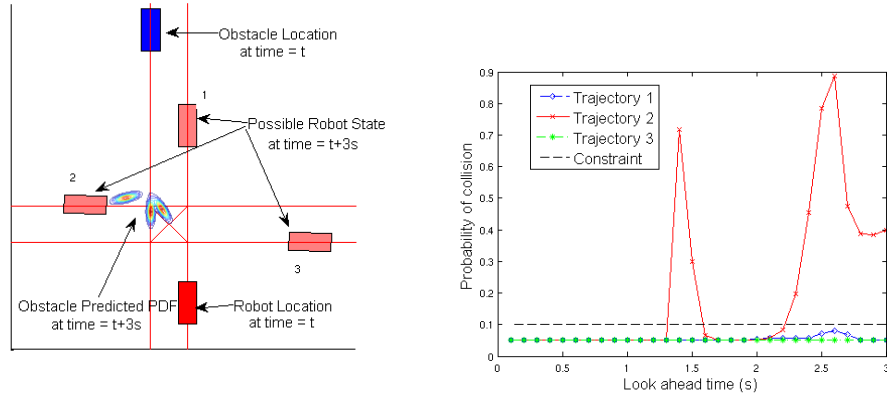
$$\begin{aligned} \mathbf{x}_{k,\text{obs}} &\sim p(\mathbf{x}_{k,\text{obs}}) \\ \mathbf{x}_{k,\text{obs}} &\sim \sum_{i=1}^{N_k} w_k^i \cdot \mathcal{N}(\mathbf{x}_{k,\text{obs}} | \mu_k^i, \Sigma_k^i) \end{aligned} \quad (5.9)$$

each mixand in the GMM can be treated independently when calculating the probability of collision, so the calculation becomes

$$p_{\text{coll}}(\mathbf{x}_{k,\text{robot}}^j, \mathbf{x}_{k,\text{obs}}) = \sum_{i=1}^{N_k} w_k^i \cdot p_{\text{coll}}(\mathbf{x}_{k,\text{robot}}^j, \mathbf{x}_{k,\text{obs}}^i) \quad (5.10)$$

where

$$\mathbf{x}_{k,\text{obs}}^i \sim \mathcal{N}(\mu_k^i, \Sigma_k^i). \quad (5.11)$$



(a) Candidate robot actions and anticipated obstacle behavior

(b) Collision probability for each candidate action over the anticipation horizon

Figure 5.4: Illustration of the safety evaluations for candidate actions. 5.4(a) shows the future robot state and the anticipated obstacle state three seconds in the future. 5.4(b) plots the likelihood of collision between the robot and the obstacle vehicle for each of the three candidate actions considered by the robot.

Although the GMM representation allows the problem of computing collision probabilities to be simplified, there is no exact method for computing the probability of collision between two rectangles given a normal distribution over their relative position. This work uses a tight upper bound on the collision probability proposed in [23]. The collision probability is computed by bloating the obstacle vehicle rectangle using its heading uncertainty, finding the combined body of the bloating obstacle rectangle and the robot rectangle, and then, after a coordinate transform, evaluating the cumulative density function of the position states of the obstacle vehicle. Accuracy can be improved, to the point of approaching the true probability of collision, by taking a sum over several discrete obstacle heading ranges rather than bloating the obstacle vehicle rectangle by its heading uncertainty.

Figure 5.4 illustrates the the anticipation of a tracked object, the calculated probability of collision for candidate robot actions, and the abstraction to Boolean environment

propositions. The robot is shown in red in Figure 5.4(a) approaching the intersection. An obstacle vehicle is shown approaching from the opposite direction. The candidate actions being considered by the robot are a left turn through the intersection, continuing straight through the intersection, and a right turn through the intersection. The GMM anticipation algorithm assumes the obstacle takes each available route with equal probability, and the pdf over its state is shown. The probability of collision for each candidate action as a function of time is plotted in Figure 5.4(b). The straight and right turn maneuvers, in this case, both have very low collision probabilities, and for the threshold shown (0.1) would be labeled as *safe*. The left turn candidate action almost guarantees a collision, and would be labeled as *unsafe*.

5.4 Analysis

The synthesized controller, discussed previously, is generated in a correct-by-construction manner, which guarantees that the resulting controller satisfies the underlying task specification if the robot’s sensors and actuators are accurate. In this paper, as in [27, 28], the probability that the robot satisfies its task is found with respect to errors in the robot’s sensors. In particular, the environmental inputs to the controller $\mathcal{X} = \{Unsafe_1, Unsafe_2, Unsafe_3\}$ will include erroneous valuations, due to errors in the obstacle anticipation and the effects of abstracting the probabilistic perception to Boolean values.

First, a set of sensor propositions $\overline{\mathcal{X}} = \{\overline{Unsafe_1}, \overline{Unsafe_2}, \overline{Unsafe_3}\}$ is created; these propositions are distinct from the environment propositions, and represent the vehicle’s observation of the state of the environment (\mathcal{X} , on the other hand, represents the *true* state of the environment). In the ideal case the values of these propositions would perfectly mimic the values of their corresponding environment propositions (i.e., $\overline{Unsafe_i}$ would be true precisely when $Unsafe_i$ is true). The use of this set of propositions allows the model to capture inaccuracies in the vehicle’s perception and the effects

of the abstraction, through false positives ($\overline{Unsafe_i}$ is true when $Unsafe_i$ is false) and false negatives ($\overline{Unsafe_i}$ is false when $Unsafe_i$ is true).

A discrete, probabilistic model of the system is then created to capture the effects of these inaccuracies. To do this, a probabilistic model of the evolution of the environment state is composed with a probabilistic model of the sensor observations and the discrete, synthesized controller. The environment model, $P(X'|X, Y)$, models changes in the environment configuration X in the next time step (denoted by the ' symbol), as a conditional probability determined by the current configurations of the environment X and robot Y ; for example, $p(Unsafe'_i) = 0.1$ indicates that the probability that route i is *unsafe* at any given time is 0.1. The sensor model, $P(\overline{X}'|X', \overline{X}, Y)$, models the new sensor configuration \overline{X}' probabilistically, conditioned on the new environment configuration X' , the current sensor configuration \overline{X} , and the current robot configuration Y . The probability $p(\overline{Unsafe'_i}|Unsafe'_i) = 0.6$, for example, indicates that the probability that the robot accurately anticipates route i to be *unsafe*, given that it actually is *unsafe*, is 0.6. The final component of the composed system model is the synthesized controller; this controller takes the form of a discrete, deterministic Finite-State Automaton (as described in Section 5.2.2).

For a detailed description of the model composition process, the reader is referred to [28]. Briefly, each state in the original automaton is adjusted to include the labeled inputs as the sensor values (in addition to the robot configuration) such that each state is now labeled with a sensor configuration \overline{X} and robot configuration Y , as determined by the synthesized controller. This state is then duplicated and augmented with each allowable environment configuration (such that $P(X'|X, Y) > 0$), and each transition is assigned a probability that is the product of $P(X'|X, Y)$ and $P(\overline{X}'|X', \overline{X}, Y)$ for the given configurations. The resulting model is a Discrete-Time Markov Chain, where each state is labeled with configurations for the environment X , the sensors \overline{X} , and the robot

Y , and all of the transitions are probabilistic.

Such a model allows for the use of off-the-shelf model checking software to find the probability that the modeled system will satisfy a temporal logic formula. More specifically, the work presented here uses the model checking software PRISM [38] to compute the probability that the model satisfies a given LTL formula. This formula is restricted to the sets of propositions used in the model (i.e., \mathcal{X} , $\overline{\mathcal{X}}$, and \mathcal{Y}), but is not confined to those formulas that are used in the task specification.

This approach can be used to find the probability that the synthesized controller will exhibit some desired behavior, given the modeled environment behavior and sensor error. Additionally, the probabilities $P(X'|X, Y)$ and $P(\overline{X}'|X', \overline{X}, Y)$ can be adjusted to quantify the effects of changes to the probabilistic models. In the work presented here, the sensor error probability is adjusted, in order to determine the effect of changing the abstraction threshold (described in Section 5.3.4).

5.5 Experimental Platform

The synthesized controller and anticipation-based safety sensors were implemented in the control algorithms of Cornell’s 2007 DARPA Urban Challenge entry, Skynet[47]. This section provides a brief overview of Skynet’s software architecture and how the synthesized controller and dynamic obstacle anticipation are incorporated, as well as a description of the integration of virtual dynamic obstacles into the car’s perceived environment.

5.5.1 Autonomous Car

Figure 5.5 provides an overview of Skynet’s software architecture. Skynet perceives its own state using a tightly coupled pose estimator that fuses GPS, inertial measurements,

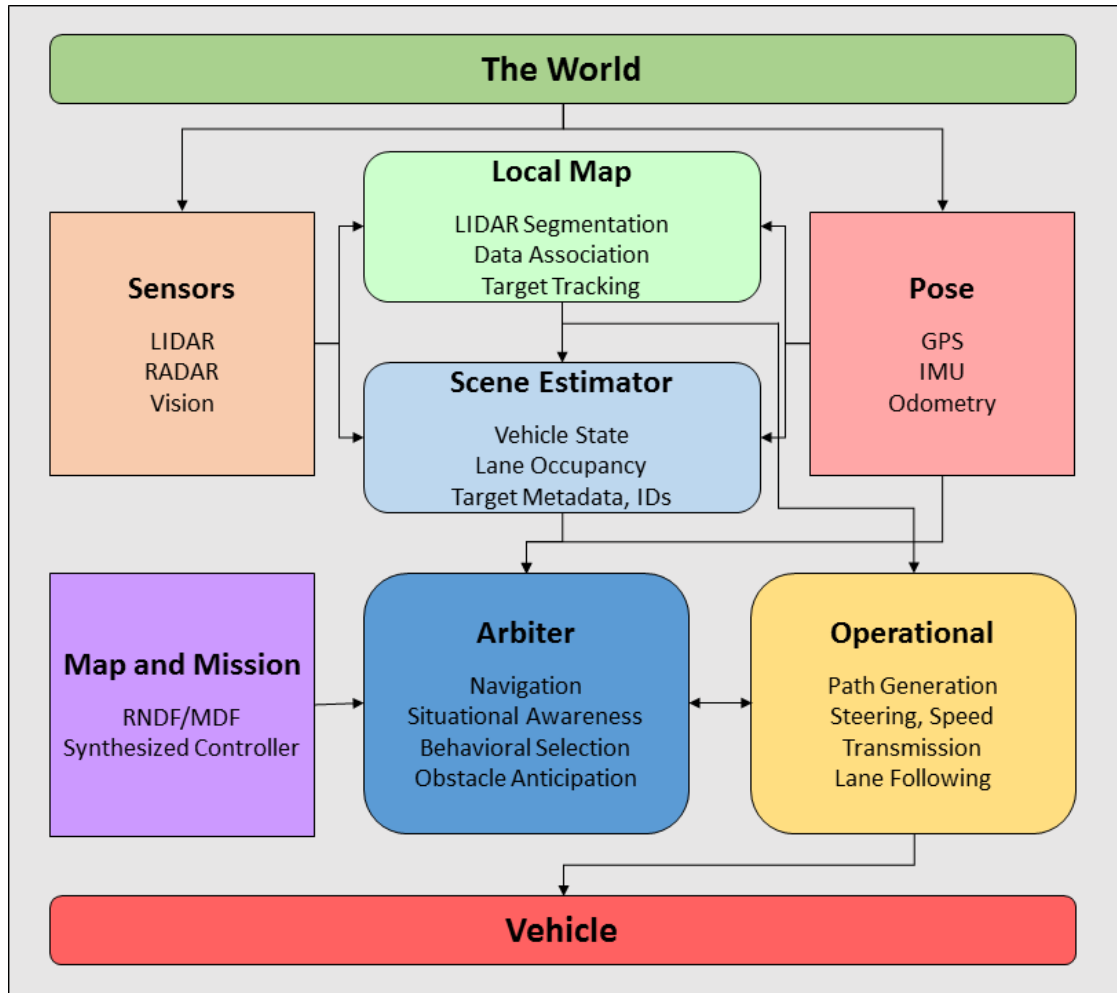


Figure 5.5: Overview of the software architecture for the autonomous car, Skynet.

and vehicle odometry data to estimate Skynet’s position in an absolute, Earth-fixed coordinate system. Skynet’s environmental perception uses a combination of RADAR and LIDAR sensors, whose returns are processed by LocalMap to arrive at “raw” clusters and tracks.

The higher-level perception is performed in Scene Estimator, which consumes Skynet’s pose estimate, raw tracked objects from Local Map, and the known map in order to develop a high-level understanding of the world suitable for planning; Skynet’s Earth-fixed pose estimate is translated to a more useful estimate of Skynet’s position within the map, and raw tracked clusters are classified, and when appropriate, also placed within

the map.

Skynet’s control algorithms are split into high-level and low-level tasks. The high-level tasks are performed by the Arbiter component, which consumes the environment estimate from Scene Estimator, the known map, and the current mission to make high-level decisions, such as choosing a direction at an intersection, making a lane change, performing a U-turn, etc. These high-level actions are executed by the low-level planner Operational, which takes the commanded action, environment estimate from Scene Estimator, known map, and current sensor returns and makes vehicle actuation commands, completing the planning loop.

Each perception and planning algorithm is implemented in C# or C++ on separate Windows computers mounted in a rack in Skynet. The algorithms communicate with each other and the vehicle over Skynet’s local network. A network of microcontrollers triggers and timestamps data from Skynet’s sensors, ensuring accurate sensor fusion even though Skynet’s perception algorithms are not implemented in hard real-time environments. A more in-depth description of Skynet’s architecture is available in Cornell’s DUC report [47].

Implementation of the synthesized controller and anticipation-based safety sensors is done entirely within Skynet’s high-level planner, Arbiter. A new PC was built and installed in Skynet to handle the high computational demands of the anticipation algorithm, discussed in more detail in previous work [25]. The primary high-level planning loop runs on its own thread, to ensure timely delivery of commands to Operational. The anticipation and collision probability calculations parallelize nicely, and are performed on separate threads. The previous complex, hand-coded high-level logic in Arbiter is replaced with a simple algorithm that steps through the synthesized high-level controller based on the results of the anticipation. Anticipation and collision checking is run constantly, not just when the state machine needs a result, and is able to anticipate the

behavior of vehicles of interest in the environment to a horizon of 4.5 seconds at greater than 3 Hz. When the state machine needs to make a transition, the transition is made based on the most recent anticipation results, which are never older than 1/3 seconds.

5.5.2 Virtual Obstacles

Because the experiment is intended to verify theoretical and simulation results that show a number of collisions, it could not be performed with other physical vehicles on the road. Instead, the dynamic obstacles used in the experiment were virtual objects, that were transmitted to, and perceived by Skynet. Each of the obstacles was controlled by a human driver who operated a simulated vehicle in an open-source driving simulation, using a driving wheel and pedals attached to a laptop computer; a picture of this setup is shown in Figure 5.6.

The position and heading of each dynamic obstacle was transmitted via wireless network to the autonomous vehicle, such that Skynet perceived the simulated cars as being physically present (as vehicle tracks in Scene Estimator), and tracked and anticipated their motion as described in Section 5.3.3. Similarly, the human drivers perceived Skynet as another actor in the computer simulation, thereby achieving realistic human driver-Skynet interactions without the risk of vehicle damage and injury in a collision. This simulation setup allowed for the operation of the experiment, without the safety and damage concerns associated with operating real cars in scenarios where collisions were expected.

5.6 Results

This section describes the resulting behavior of the autonomous vehicle operating in the scenario presented in Section 5.3.1. The behavior is evaluated with respect to the safety



Figure 5.6: Picture of the setup for the dynamic obstacles. A driving wheel was attached to a laptop computer, which was running an open-source driving simulator that was modified to show to the positions of all of the obstacle vehicles and Skynet.

of the autonomous vehicles operation (avoiding collisions with other vehicles), and the results of a formal analysis are compared with experimental data.

5.6.1 Formal Analysis

As discussed in Section 5.4, a probabilistic model of the system was created and used to formally verify properties of the system, using the PRISM model checking software. To get the probabilities used in the formal model, $P(X'|X, Y)$ and $P(\overline{X}'|X', \overline{X}, Y)$, statistical data was extracted from a high-fidelity simulation of the scenario. This simulation used dynamic obstacles with simple waypoint-following controllers (with no avoidance logic), and tracked the positions of the obstacles and the autonomous car. Additionally, the decisions made by the autonomous vehicle were stored for each intersection, along with the calculated collision probabilities (from the obstacle anticipation). This information is used to calculate the model probabilities for the environment and sensor models.

The environment model $P(X'|X, Y)$ describes the probability that, when the vehicle arrives at any intersection, each of the possible routes is unsafe. These probabilities were modeled as independent of the previous value or the location of the robot, and were found to be:

$$p(Unsafe'_1) = 0.06490$$

$$p(Unsafe'_2) = 0.12876$$

$$p(Unsafe'_3) = 0.23636$$

These probabilities were found by computing the fraction of each choice $Drive_1$, $Drive_2$, and $Drive_3$ (which correspond to the $Unsafe_1$, $Unsafe_2$, and $Unsafe_3$, propositions, respectively), that resulted in a collision when taken by the autonomous vehicle. That is, $p(Unsafe'_i)$ was found by dividing the number of $Drive_i$ choices that resulted in a

collision by the number of $Drive_i$ choices that were taken, in total. These probabilities were obtained from simulation data of 1848 intersection traversals, and provide a more accurate model than treating all of the $Unsafe_i$ sensors identically (i.e., letting $p(Unsafe'_1) = p(Unsafe'_2) = p(Unsafe'_3)$ and determining the single value from all decisions made).

The sensor error model $P(\overline{X}'|X', \overline{X}, Y)$ was constructed similarly, though it is important to note that these probabilities also change as the abstraction threshold is adjusted. In this case, the error model was used to model the probability that the abstracted sensor correctly determined when a road was *unsafe* $p(\overline{Unsafe}'_i|Unsafe'_i)$, and the probability that the abstracted sensor correctly determined when a road was *safe* $p(\neg\overline{Unsafe}'_i|\neg Unsafe'_i)$, for each of the possible choices $i \in \{1, 2, 3\}$. These probabilities were modeled to be dependent only on the value of their corresponding environment proposition, and independent of the previous sensor values or the robot's location. Because the sensor information is obtained by abstracting the probabilistic anticipation to a Boolean value for the sensor, these model probabilities are also dependent on the threshold value used in that abstraction (e.g., if the threshold line in Figure 5.4(b) is lowered to ≈ 0.075 then trajectory 1, in addition to trajectory 2, becomes *unsafe*).

Figure 5.7 shows the sensor model probabilities as a function of the abstraction threshold. As with the environment model probabilities, these probabilities were calculated from simulation data (from the same 1848 intersections used to determine the environment probabilities). Unlike the environment model probabilities, and because the anticipation and abstraction is performed identically for each sensor, the sensor performance was modeled such that all of the sensors (\overline{Unsafe}_1 , \overline{Unsafe}_2 , and \overline{Unsafe}_3) have the same probabilities. In this case, for each choice taken by the autonomous vehicle, the abstracted value of the corresponding sensor was compared to the result of the choice. That is, $p(\overline{Unsafe}'_i|Unsafe'_i)$ was found, for a given threshold value, by

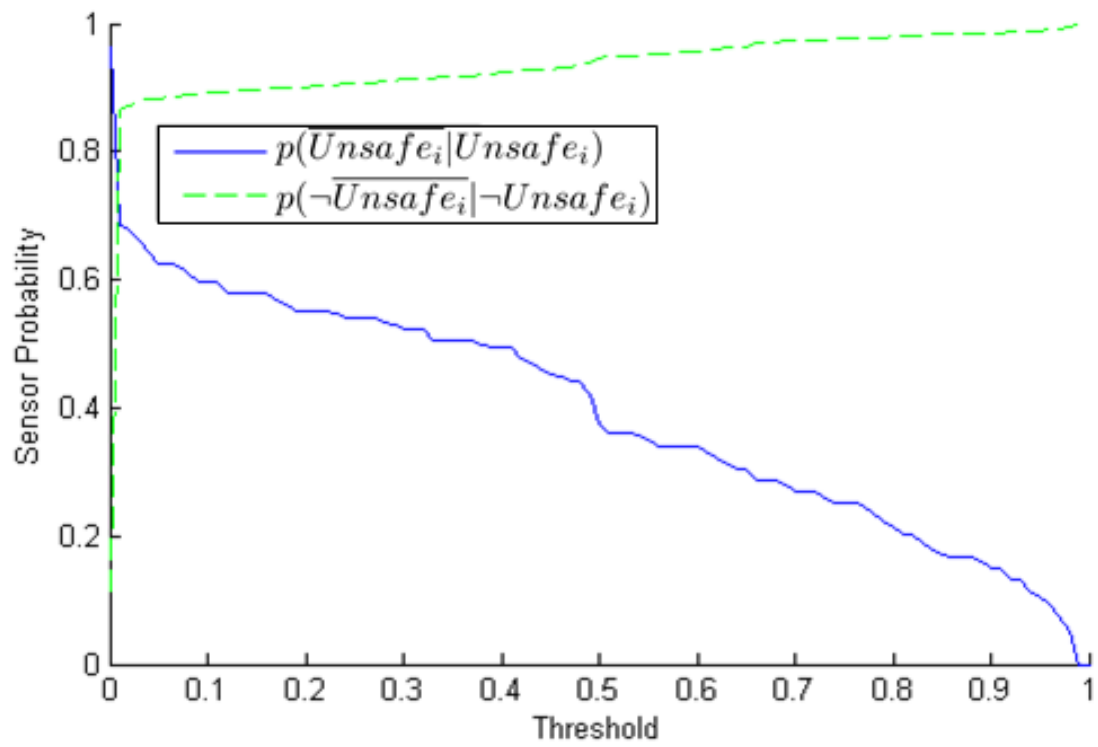


Figure 5.7: Sensor probability models, as a function of the abstraction threshold.

dividing the number of choices taken that were *unsafe* (resulted in a collision) and were deemed so by the abstracted threshold, by the total number of choices taken that were *unsafe*. The same process was used to find $p(\neg \overline{Unsafe_i}' | \neg Unsafe_i')$: the number of choices taken that were *safe* (no collision) and were deemed so by the abstracted threshold, was divided by the total number of choices taken that were *safe*.

Note that, as the threshold (collision probability at which a road is considered unsafe) increases, the vehicle's behavior becomes more aggressive. This can be seen in Figure 5.7, where the abstracted sensor becomes more likely to consider a road to be *safe*, and less likely to consider a road to be *unsafe*. Additionally, there is an obvious jump that occurs at the smallest non-zero threshold (0.01) shown in the figure. This is because the probabilistic anticipation represents the states of the obstacles with GMMs; as a result, the calculated probability of collision will always be greater than 0 (even if it is very small) and the abstracted sensor, at a threshold of 0, will determine all possible choices to be *unsafe*. The resulting model is such that $p(\overline{Unsafe_i}') = 1$ and $p(\neg \overline{Unsafe_i}') = 0$, regardless of the environment proposition. Additionally, this figure shows abrupt changes in the probabilities (e.g., around the threshold of 0.5); this is likely due to the repetition of discrete events in the scenario (the discrete choices made by the dynamic obstacles), where a small change in the abstraction threshold may alter the vehicle's behavior in multiple events.

Given this probabilistic model of the environment and the sensors, a model of the discrete system can be composed (as briefly discussed in Section 5.4), and a model checker can be used to compute the probability that the model satisfies a given LTL property. Figure 5.8 shows the probability that the vehicle safely (without colliding with a dynamic obstacle) traverses 8, 16, and 32 intersections, as a function of the abstraction threshold value. This is found as the probability that the composed model satisfies the LTL formula $\Box \wedge_{i \in \{1,2,3\}} (Unsafe_i \rightarrow \neg Drive_i)$, within a bounded number of intersec-

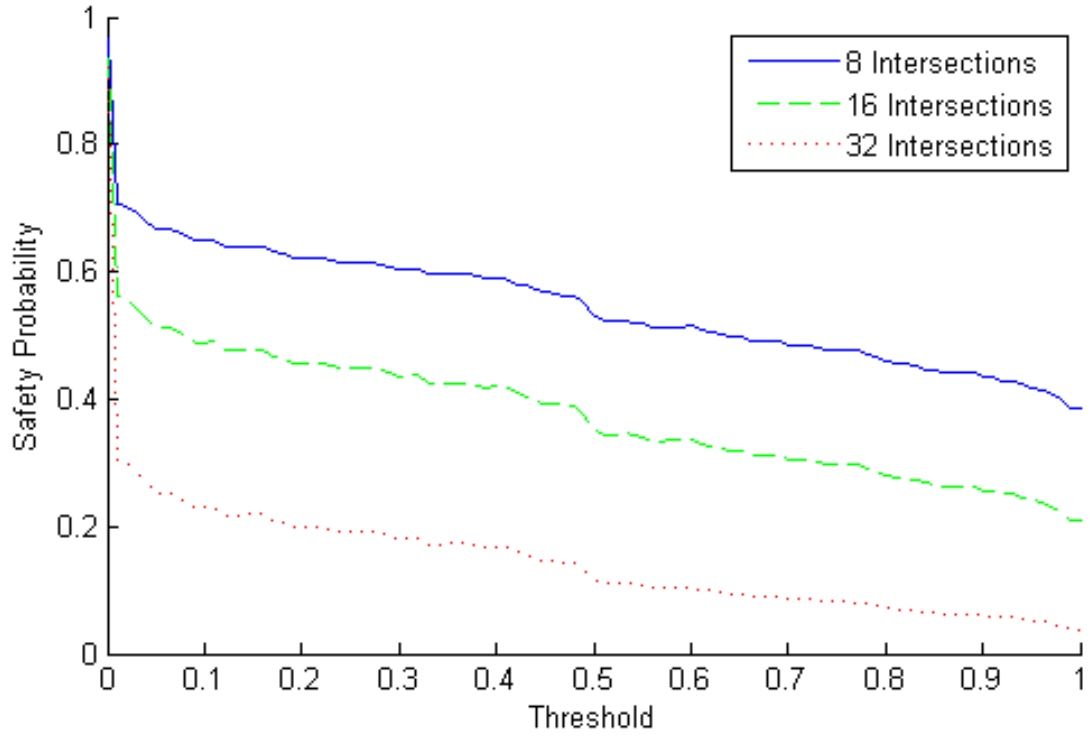


Figure 5.8: Probability that the autonomous vehicle satisfies $\Box \wedge_{i \in \{1,2,3\}} (Unsafe_i \rightarrow \neg Drive_i)$ bounded by the number of traversed intersections, and found as a function of the abstraction threshold.

tion traversals. For Figure 5.8, as well as Figure 5.7, the probabilities are calculated for a range of threshold values from 0 to 1, by increments of 0.01.

As would be expected, as the bound on the number of intersections increases, the safety probability decreases. Additionally, as the abstraction threshold increases and the vehicle drives more aggressively, the probability that it safely traverses all of its intersections decreases. As with Figure 5.7, this figure shows an abrupt change as the threshold becomes non-zero; at a threshold of 0 the abstracted sensor will always determine all routes to be *unsafe*, and the vehicle will remain stopped. As a result, the vehicle will satisfy the LTL formula $\Box \wedge_{i \in \{1,2,3\}} (Unsafe_i \rightarrow \neg Drive_i)$ with a probability of 1.0 (since $Drive_i$ will always be false).



Figure 5.9: Vehicle roadmap, superimposed on a picture of the parking lot where the experiment was conducted. Photo source: Google Earth

5.6.2 Experiment

To validate the formal analysis, an experiment was conducted to find the statistical likelihood that the vehicle will safely complete a run. This experiment used the full-scale autonomous vehicle, running a synthesized controller to make discrete decisions when arriving at intersections. The vehicle operated on the roadmap shown in Figure 5.3, navigating a sufficiently empty parking lot in the imposed lanes. A picture of the lanes, superimposed on an overhead photograph of the parking lot, is shown in Figure 5.9. During operation, the vehicle tracked and predicted the motion of three dynamic obstacles operating on the same road system.

All three obstacles were controlled by human drivers, where each of the drivers was given a prescribed path to follow, and told to drive as if they had the right of way at each intersection, and to actively avoid collisions with the autonomous vehicle only in

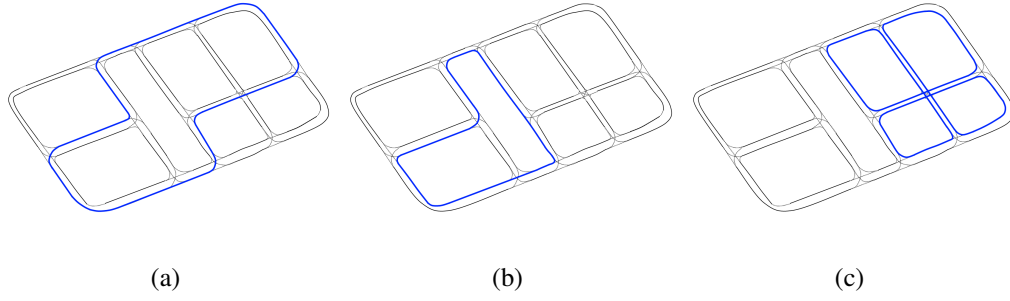


Figure 5.10: The three different paths for the obstacle vehicles, as driven by different operators in a driving simulation.

Abstraction Threshold	Collision-Free Runs	Total Runs	Percentage
0.25	8	9	0.8889
0.50	6	8	0.7500
0.75	7	11	0.6364

Table 5.1: Experiment results for 3 different abstraction threshold values.

the case where they were driving behind it in the same lane (i.e., not to rear-end the autonomous vehicle). The three different paths for these drivers are shown in Figure 5.10.

For the experiment, all three obstacle vehicles and the autonomous vehicle drove around the roadmap continuously, while the autonomous vehicle recorded its own pose and that of the obstacle vehicles, as well as the decisions made at each intersection. The data was then split into runs based on the 4 goal segments shown in Figure 5.3; each run consisted of the autonomous vehicle driving from one of the goal regions to the next goal (in numerical order). Split this way, each run consisted of precisely 8 intersections, where the autonomous vehicle queried the synthesized controller to make a decision regarding the next road segment. Table 5.1 shows the results (the number of collision-free runs, the total number of runs, and the percentage of runs that were collision-free) for three different values of the abstraction threshold.

As the abstraction threshold increases, the autonomous vehicle allows a higher probability of collision with the obstacles to be considered *safe*. As a result, the higher abstraction thresholds yield a smaller percentage of the total runs without collisions.

That is, the experimental data shows that, as the autonomous vehicle drives more aggressively, it becomes less likely to complete its run without colliding with an obstacle vehicle.

5.6.3 Results comparison

Figure 5.11 shows a comparison of the formal analysis results with the experimental data. The formal analysis results show the probability that the vehicles satisfies the formula $\Box \wedge_{i \in \{1,2,3\}} (Unsafe_i \rightarrow \neg Drive_i)$ for a time-bound of 8 intersection traversals; the experimental results show the percentage of runs (each consisting of 8 intersection traversals) during which the autonomous vehicle successfully avoided collisions with the obstacle vehicles. Additionally, the experimental results are shown with 95% confidence bounds on the percentage of collision-free runs. The figure also shows the resulting probabilities of a formal analysis (at thresholds of 0.25, 0.5, and 0.75) for a probabilistic model of the system where the environment and sensor probabilities are obtained statistically from the experimental data, rather than the simulation data. Due to the limited amount of experimental data available, the model was evaluated only at these three thresholds. In order to evaluate the model for the full range of thresholds, more experimental data would need to be collected, explicitly for this purpose (the vehicle would need to be set to ignore the calculated collision probability when making decisions, such that the accuracy of the sensors could be evaluated even at high collision probabilities).

As shown in Figure 5.11, the experimental data yields results that follow the same trend as the formal analysis results (where a higher threshold yields a lower safety probability), though the experiment results show a higher safety probability than the formal analysis results at the corresponding thresholds. This difference can be due a number of different factors.

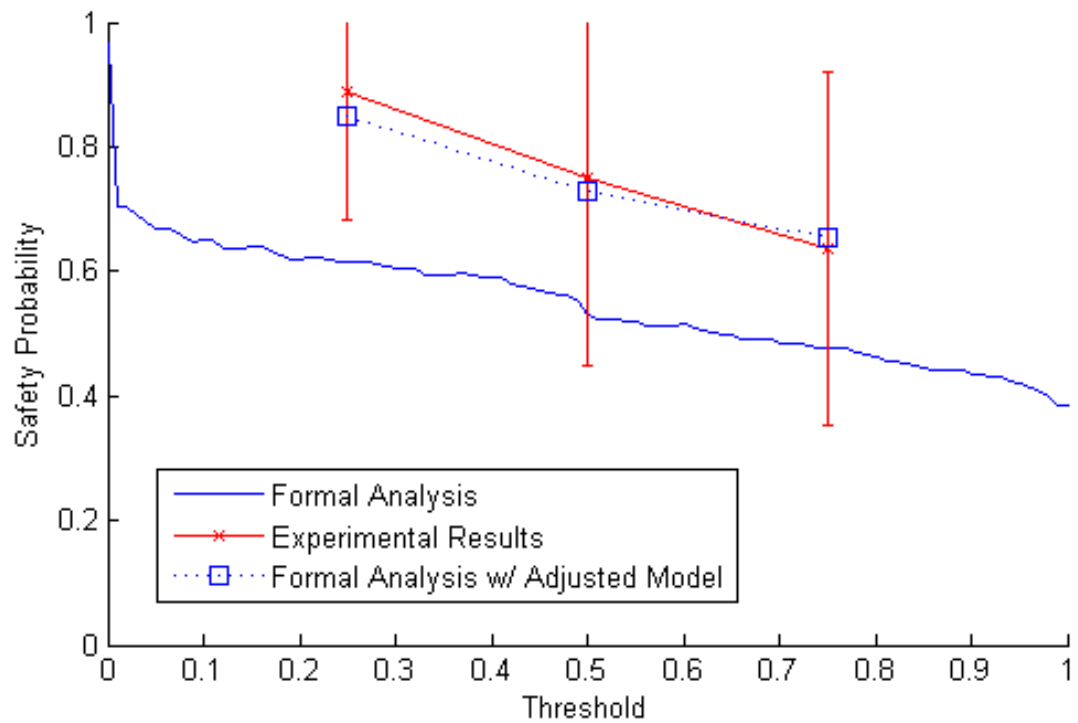


Figure 5.11: Comparison of the results for the formal analysis with the experimental results (with 95% confidence intervals). The results of a formal analysis where the model probabilities were obtained from the experimental data (rather than simulation data) is also shown.

One likely cause is the behavior of the human drivers. The model probabilities used in the formal analysis (which are shown in Figure 5.7) were obtained from a simulated system, where the obstacle vehicles were driven by simple waypoint-following behaviors. By contrast, the experiment was conducted with human drivers controlling the obstacle vehicles; while the human drivers were instructed to behave as if they always had the right-of-way, it is reasonable to expect that their natural reactions to close encounters with other vehicles would lead them to actively avoid collisions in some scenarios where a waypoint-following controller would not. In fact, if the experimental data is used to obtain the statistics for the probabilities in the formal model, an analysis of the model gives results that are more closely aligned with the experimental results, as shown in Figure 5.11. At an abstraction threshold of 0.25, analysis of the formal model obtained from experiment data yields a safety probability of 0.8506; for abstraction thresholds of 0.5 and 0.75, the safety probabilities are 0.7300 and 0.6544, respectively. These values align much more closely with the experimental data, though it is important to note that considerably less experimental data was used to find the model statistics than when they were generated from simulation data.

Additionally, the experimental results shown in the figure were obtained from a small number of samples. Due to the small quantity of data available, the actual values of the safety probabilities are relatively uncertain (as evidenced by the large spread of the confidence intervals in Figure 5.11). Results obtained from a larger quantity of data may be closer to the formal analysis results.

5.7 Conclusions

This paper describes a method for interpreting the probabilistic anticipation of dynamic obstacles as abstract sensor values for a synthesized, correct-by-construction controller. The approach is applied to the problem of a full-scale autonomous vehicle operating in

an environment where it must avoid other vehicles; the approach is implemented and running in real-time on Cornell's 2007 DARPA Urban Challenge entry vehicle. During execution, the motion of the dynamic obstacles are anticipated, and used to calculate the expected probability of collision between the obstacles and the autonomous vehicle. If this value is above a specified threshold, the vehicle's route is deemed *unsafe*, and the high-level controller, which was synthesized from a task specification that required that the vehicle only drive on roads that were *safe*, directs the vehicle to avoid that route.

The formal analysis of a discrete model of the system, as well as experimental data from execution of the autonomous car, illustrates the effects of changes in the sensor abstraction threshold on the behavior of the autonomous vehicle. Both the formal analysis and the experimental results show that a higher threshold value results in more aggressive behavior for the vehicle, and a lower probability of safely completing a run. Furthermore, the agreement between the experimental data and the formal results suggest that the system model and analysis are viable methods for assessing the behavior of an autonomous system operating with a synthesized controller.

BIBLIOGRAPHY

- [1] D. Alspach and HW Sorenson. Nonlinear Bayesian Estimation using Gaussian Sum Approximations. *Automatic Control, IEEE Transactions on*, 17(4):439–448, 1972.
- [2] M. Althoff and J.M. Dolan. Online Verification of Automated Road Vehicles Using Reachability Analysis. *Robotics, IEEE Transactions on*, 30(4):903–918, Aug 2014.
- [3] Andrew Bacha, Cheryl Bauman, Ruel Faruque, Michael Fleming, Chris Terwelp, Charles Reinholtz, Dennis Hong, Al Wicks, Thomas Alberi, David Anderson, et al. Odin: Team VictorTango’s Entry in the DARPA Urban Challenge. *Journal of Field Robotics*, 25(8):467–492, 2008.
- [4] A. Bhatia, L.E. Kavraki, and M.Y. Vardi. Sampling-Based Motion Planning with Temporal Goals. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2689–2696. IEEE, 2010.
- [5] Mark Campbell, Magnus Egerstedt, Jonathan P How, and Richard M Murray. Autonomous Driving in Urban Environments: Approaches, Lessons and Challenges. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 368(1928):4649–4672, 2010.
- [6] J.S. Choi, G. Eoh, J. Kim, Y. Yoon, J. Park, and B.H. Lee. Analytic Collision Anticipation Technology Considering Agents’ Future Behavior. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1656–1661. IEEE, 2010.
- [7] I. Cizelj and C. Belta. Negotiating the Probabilistic Satisfaction of Temporal Logic

- Motion Specifications. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, November 2013.
- [8] D.C. Conner, H. Kress-Gazit, H. Choset, and A.A. Rizzi. Valet Parking Without a Valet. *Proc. of IEEE/RSJ*, pages 572–577, oct 2007.
 - [9] C. Courcoubetis and M. Yannakakis. The Complexity of Probabilistic Verification. *Journal of ACM*, 42(4):587–907, Jul 1995.
 - [10] X. C. Ding, S. L. Smith, C. Belta, and D. Rus. LTL Control in Uncertain Environments with Probabilistic Satisfaction Guarantees. In *18th IFAC World Congress*, 2011.
 - [11] X. C. Ding, J. Wang, M. Lahijanian, I. Ch. Paschalidis, and C. A. Belta. Temporal Logic Motion Control Using Actor-Critic Methods. In *2012 IEEE International Conference on Robotics and Automation*, pages 4687–4692. IEEE, May 2012.
 - [12] N.E. Du Toit and J.W. Burdick. Robotic Motion Planning in Dynamic, Cluttered, Uncertain Environments. In *2010 IEEE Conference on Robotics and Automation*, pages 966–973, 2010.
 - [13] Noel E Du Toit and Joel W Burdick. Robot Motion Planning in Dynamic, Uncertain Environments. *Robotics, IEEE Transactions on*, 28(1):101–115, 2012.
 - [14] E.A. Emerson. Temporal and Modal Logic. *Handbook of Theoretical Computer Science*, E(16):995–1072, July 1990.
 - [15] G.E. Fainekos. Revising Temporal Logic Specifications for Motion Planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 40–45. IEEE, 2011.

- [16] G.E. Fainekos, A. Girard, and G.J. Pappas. Hierarchical Synthesis of Hybrid Controllers from Temporal Logic Specifications. In *Hybrid Systems: Computation and Control*, pages 203–216. Springer, 2007.
- [17] G.E. Fainekos, H. Kress-Gazit, and G.J. Pappas. Hybrid Controllers for Path Planning: A Temporal Logic Approach. *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 4885–4890, 2006.
- [18] C. Finucane, G. Jing, and H. Kress-Gazit. LTLMoP : Experimenting with Language , Temporal Logic and Robot Control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems.*, pages 1988–1993, Taipei, Taiwan, 2010.
- [19] L. Fletcher, S. Teller, E. Olson, D. Moore, Y. Kuwata, J. How, J. Leonard, I. Miller, M. Campbell, D. Huttenlocher, et al. The MIT-Cornell Collision and Why it Happened. *Journal of Field Robotics*, 25(10):775–807, 2008.
- [20] Jie Fu and Ufuk Topcu. Probably Approximately Correct MDP Learning and Control With Temporal Logic Constraints. In *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014.
- [21] C. Fulgenzi, A. Spalanzani, and C. Laugier. Probabilistic Rapidly-Exploring Random Trees for Autonomous Navigation Among Moving Obstacles. In *Workshop on safe navigation, IEEE International Conference on Robotics and Automation (ICRA)*, 2009.
- [22] E.M. Hahn, H. Hermanns, and L. Zhang. Probabilistic Reachability for Parametric Markov Models. *International Journal on Software Tools for Technology Transfer*, 13:3–19, April 2010.
- [23] Jason Hardy and Mark Campbell. Contingency Planning Over Probabilistic Ob-

- stacle Predictions for Autonomous Road Vehicles. *Robotics, IEEE Transactions on*, 29(4):913–929, 2013.
- [24] Jason Hardy, Frank Havlak, and Mark Campbell. Multiple-Step Prediction Using a Two Stage Gaussian Process Model. In *American Control Conference (ACC)*, 2014, pages 3443–3449. IEEE, 2014.
- [25] F. Havlak and M. Campbell. Discrete and Continuous, Probabilistic Anticipation for Autonomous Robots in Urban Environments. *Robotics, IEEE Transactions on*, 30(2):461–474, April 2014.
- [26] G. Jing, R. Ehlers, and H. Kress-Gazit. Shortcut Through an Evil Door: Optimality of Correct-By-Construction Controllers in Adversarial Environments. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 4796–4802, 2013.
- [27] B. Johnson, F. Havlak, M. Campbell, and H. Kress-Gazit. Execution and Analysis of High-Level Tasks with Dynamic Obstacle Anticipation. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2012.
- [28] B. Johnson and H. Kress-Gazit. Probabilistic Analysis of Correctness of High-Level Robot Behavior with Sensor Error. In *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.
- [29] B. Johnson and H. Kress-Gazit. Probabilistic Guarantees for High-Level Robot Behavior in the Presence of Sensor Error. *Autonomous Robots*, 33:309–321, 2012.
- [30] B. Johnson and H. Kress-Gazit. Analyzing and Revising High-Level Robot Behaviors Under Actuator Uncertainty. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, November 2013.

- [31] Simon J Julier and Jeffrey K Uhlmann. A New Extension of the Kalman Filter to Nonlinear Systems. In *Int. symp. aerospace/defense sensing, simul. and controls*, pages 3–2. Orlando, FL, 1997.
- [32] S. Karaman and E. Frazzoli. Sampling-Based Motion Planning with Deterministic μ -Calculus Specifications. In *IEEE Conference on Decision and Control*, pages 2222–2229, 2009.
- [33] K. Kim, G.E. Fainekos, and S. Sankaranarayanan. On the Revision Problem of Specification Automata. In *Proceedings of the IEEE Conference on Robotics and Automation*, 2012.
- [34] M. Kloetzer and C. Belta. A Fully Automated Framework for Control of Linear Systems from LTL Specifications. In *Hybrid Systems: Computation and Control*, volume 3927, pages 333–347, 2006.
- [35] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. From Structured English to Robot Motion. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 2717–2722. Citeseer, Oct 2007.
- [36] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. Where’s Waldo? Sensor-Based Temporal Logic Motion Planning. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3116–3121. Citeseer, April 2007.
- [37] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. Temporal-Logic-Based Reactive Mission and Motion Planning. *Robotics, IEEE Transactions on*, 25(6):1370–1381, dec 2009.
- [38] M. Kwiatkowska, G. Norman, and Parker D. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *Proc. 23rd International Conference on Computer*

Aided Verification (CAV'11), volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

- [39] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic Symbolic Model Checker. In P. Kemper, editor, *Proc. Tools Session of Aachen 2001 International Multiconference on Measurement, Modeling and Evaluation of Computer-Communication Systems*, pages 7–12, September 2001.
- [40] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic Model Checking. *Formal Methods for Performance Evaluation*, pages 220–270, 2007.
- [41] M. Kwiatkowska, G. Norman, D. Parker, and M. Kattenbelt. PRISM Probabilistic Model Checker, 2010.
- [42] M. Lahijanlian, S.B. Andersson, and C. Belta. A Probabilistic Approach for Control of a Stochastic System from LTL Specifications. In *IEEE Conference on Decision and Control*, pages 2236–2241, Shanghai, P.R. China, 2009.
- [43] M. Lahijanlian, J. Wasniewski, S.B. Andersson, and C. Belta. Motion Planning and Control from Temporal Logic Specifications with Probabilistic Satisfaction Guarantees. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3227–3232, Anchorage, AK, May 2010.
- [44] S.G. Loizou and K.J. Kyriakopoulos. Automatic Synthesis of Multi-Agent Motion Tasks Based on LTL Specifications. In *IEEE Conference on Decision and Control*, pages 153–158 Vol.1. IEEE, 2004.
- [45] D.M. Lyons, R.C. Arkin, P. Nirmal, S. Jiang, T-M Liu, and J. Deeb. Getting it Right the First Time: Robot Mission Guarantees in the Presence of Uncertainty. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, November 2013.

- [46] A. I. Medina Ayala, S. B. Andersson, and C. Belta. Probabilistic Control from Time-Bounded Temporal Logic Specifications in Dynamic Environments. In *2012 IEEE International Conference on Robotics and Automation*, pages 4705–4710. IEEE, May 2012.
- [47] Isaac Miller, Mark Campbell, Dan Huttenlocher, Frank-Robert Kline, Aaron Nathan, Sergei Lupashin, Jason Catlin, Brian Schimpf, Pete Moran, Noah Zych, et al. Team Cornell’s Skynet: Robust Perception and Planning in an Urban Environment. *Journal of Field Robotics*, 25(8):493–527, 2008.
- [48] T. Ohki, K. Nagatani, and K. Yoshida. Collision Avoidance Method for Mobile Robot Considering Motion and Personal Spaces of Evacuees. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1819–1824. IEEE, 2010.
- [49] P. Peterson and F. Johansson. SympyCore - an Efficient Pure Python Computer Algebra System, 2011.
- [50] Stephane Petti and Thierry Fraichard. Safe Motion Planning in Dynamic Environments. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 2210–2215. IEEE, 2005.
- [51] N. Piterman, A. Pnueli, and Y. Sa’ar. Synthesis of Reactive (1) Designs. *Verification, Model Checking, and Abstract*, pages 364–380, 2006.
- [52] V. Raman and H. Kress-Gazit. Towards Minimal Explanations of Unsynthesizability for High-Level Robot Behaviors. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, November 2013.
- [53] K. Y. Rozier. Linear Temporal Logic Symbolic Model Checking. *Computer Science Review*, 2010.

- [54] S.L. Smith, J. Tůmová, C. Belta, and D. Rus. Optimal Path Planning Under Temporal Logic Constraints. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3288–3293, Oct 2010.
- [55] P. Tabuada and G.J. Pappas. Model Checking LTL Over Controllable Linear Systems is Decidable. *Hybrid Systems: Computation and Control*, pages 498–513, 2003.
- [56] P. Tabuada and G.J. Pappas. Linear Time Logic Control of Discrete-Time Linear Systems. *Automatic Control, IEEE Transactions on*, 51(12):1862–1877, Dec 2006.
- [57] A. Ulusoy, T. Wongpiromsarn, and C. Belta. Incremental Control Synthesis in Probabilistic Environments with Temporal Logic Constraints. In *Decision and Control (CDC), 2012 IEEE Conference on*, pages 7658–7663, 2012.
- [58] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher R. Baker, Robert Bittner, M. N. Clark, John M. Dolan, Dave Duggins, Tugrul Galatali, Christopher Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas M. Howard, Sascha Kolski, Alonzo Kelly, Maxim Likhachev, Matthew McNaughton, Nick Miller, Kevin Peterson, Brian Pilnick, Raj Rajkumar, Paul E. Rybski, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod Snider, Anthony Stentz, William Whitaker, Ziv Wolkowicki, Jason Ziglar, Hong Bae, Thomas Brown, Daniel Demitrish, Bakhtiar Litkouhi, Jim Nickolaou, Varsha Sadekar, Wende Zhang, Joshua Struble, Michael Taylor, Michael Darms, and Dave Ferguson. Autonomous Driving in Urban Environments: Boss and the Urban Challenge. *J. Field Robotics*, 25(8):425–466, 2008.
- [59] E.M. Wolff, U. Topcu, and R.M. Murray. Optimal Control with Weighted Average

Costs and Temporal Logic Specifications. In *Robotics: Science and Systems (RSS)*, 2012.

- [60] Eric M Wolff, Ufuk Topcu, and Richard M Murray. Optimization-based Trajectory Generation with Linear Temporal Logic Specifications. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014. To appear.
- [61] T. Wongpiromsarn, U. Topcu, and R.M. Murray. Receding Horizon Temporal Logic Planning. In *IEEE Conference on Decision and Control (CDC)*, pages 5997–6004. IEEE, Dec 2009.
- [62] T. Wongpiromsarn, U. Topcu, and R.M. Murray. Automatic Synthesis of Robust Embedded Control Software. *AAAI Spring Symposium on Embedded Reasoning: Intelligence in Embedded Systems*, 2010.
- [63] B.D. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J.A. Bagnell, M. Hebert, A.K. Dey, and S. Srinivasa. Planning-Based Prediction for Pedestrians. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 3931–3936. IEEE, 2009.